君正®

Magik 训练量化使用指南

Date: Feb. 2022

君正®

Magik训练量化
使用指南

Release history

Date	Author	Revision	Change
Feb. 2022	Heidi	1.0.1	First release

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10
Xibeiwang East Road, Haidian District, Beijing, China,
Tel: 86-10-56345000
Fax:86-10-56345001
Http: //www.ingenic.com

目录

1. 简介

本指南主要针对使用君正处理芯的 Magik 平台的新手，这里以 pytorch 框架下的 Yolov5s 的 person 为例(target_devic 为 T40)，从环境搭建、数据准备、网络训练、模型转换到最终的上板运行进行全流程的详细介绍，旨在引导使用者熟悉 Magik 的使用方法，进行实现上板流程。

2. 环境安装要求

- Linux
- GCC (>=5.4.0)
- Nvidia Driver
- CUDA (>=9.0)
- CUDNN
- Python (>=3.5)
- Pytorch (>=1.3)
- Torchvision

若上述环境均已具备(版本可不做固定要求)，可直接转到步骤 3.9。

3. 环境安装步骤

3.1 Ubuntu

- 装机要求：Ubuntu16.04

3.2 GCC

- 版本要求: 5.4.0
- 具体步骤

  1. 查看当前系统 Ubuntu16.04 的原装 GCC 版本：

  $ gcc -v

  2. 下载: http://ftp.gnu.org/gnu/gcc

  3. 编译安装：

  $ tar -zxvf gcc-5.4.0.tar.bz2

  $ cd gcc-5.4.0

  $ ./contrib/download_prerequisites

  $ cd .. ; mkdir gcc-build-5.4.0

  $ cd gcc-build-5.4.0

$ ../gcc-5.4.0/configure --enable-checking=release –enable-languages=c,c++ --disable-multilib

$ sudo make

$ sudo make install

4. 检查：

再次 gcc -v 查看版本

3.3 Nvidia Driver

- 版本要求：>= 440
- 具体步骤

1．下载驱动程序

英伟达官网地址：

http://www.nvidia.cn/Download/index.aspx?lang=cn

2．禁用 nouveau 第三方驱动

打开编辑配置文件：

$ sudo gedit /etc/modprobe.d/blacklist.conf

在最后一行添加：blacklist nouveau

改好后执行命令：$ sudo update-initramfs -u

重启使之生效：$ reboot

3．安装驱动

执行命令：$ lsmod | grep nouveau

禁用 X 服务：$ sudo /etc/init.d/lightdm stop (或者：sudo service lightdm stop)

给驱动 run 文件赋予可执行权限：$ sudo chmod a+x NVIDIA-Linux-x86_64-440.64.run （下载的驱动文件名）

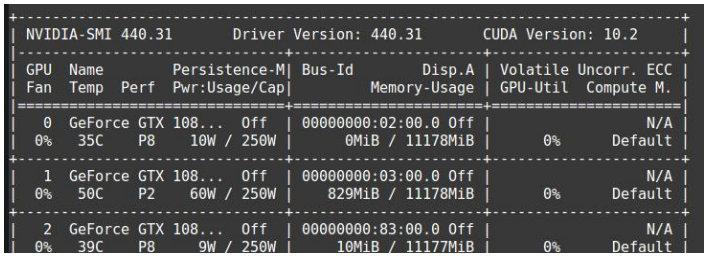
安装：$ sudo ./NVIDIA-Linux-x86_64-440.64.run -no-opengl-files

开启 X 服务：sudo /etc/init.d/lightdm start (或者：sudo service lightdm start) –no-opengl-files 只安装驱动文件，不安装 OpenGL 文件。这个参数最重要 –no-x-check 安装驱动时不检查 X 服务，–no-nouveau-check 安装驱动时不检查 nouveau，后面两个参数可不加。

4．检查

重启，没有问题，输入命令：$ nvidia-smi

如果出现了驱动版本就表示安装成功了。



3.4 CUDA

- 版本要求：10.0
- 具体步骤

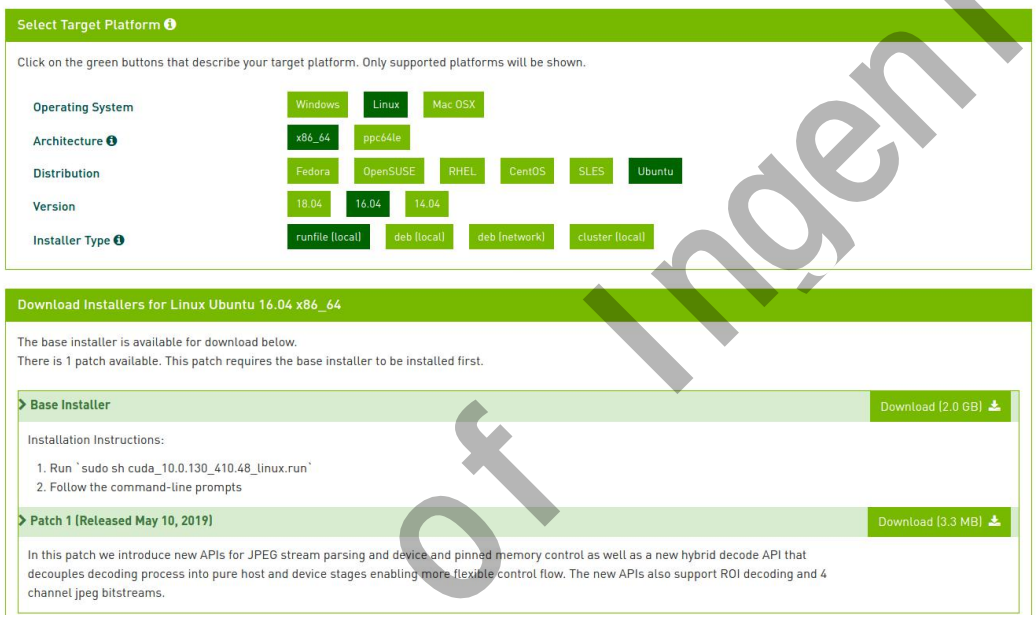
  1．下载：

  登录官网 https://developer.nvidia.com/cuda-10.0-download-archive?

  选择 linux—x86_64—Ubuntu—16.04—runfile(local)

  下载 cuda_10.0.130_410.48_linux.run

CUDA Toolkit 10.0 Archive



  2．安装：

  $ sudo sh cuda_10.0.130_410.48_linux.run

  3．查看版本：

  $ cat /usr/local/cuda/version.txt

  4．cuda 设置(适用于多个不同版本时)：

  $ export PATH=/usr/local/cuda/bin:$PATH

  $ export LD_LIBRARY_PATH=

  /usr/local/cuda/lib64:$LD_LIBRARY_PATH

3.5 CUDNN

- 版本要求：7.6.5
- 具体步骤

  1．下载：

  登陆官网： https://developer.nvidia.com/rdp/cudnn-download

  选择 cuDNN Library for Linux：cudnn-10.0-linux-x64-v7.6.5.32.tgz

2．解压：

$ tar -zxvf cudnn-10.0-linux-x64-v7.6.5.32.tgz

得到：

cuda/include/cudnn.h

cuda/NVIDIA_SLA_cuDNN_Support.txt

cuda/lib64/libcudnn.so

cuda/lib64/libcudnn.so.7

cuda/lib64/libcudnn.so.7.6.5

cuda/lib64/libcudnn_static.a

3．拷贝：

$ sudo cp cuda/lib64/* /usr/local/cuda-10.0/lib64/

$ sudo cp cuda/include/* /usr/local/cuda-10.0/include/

4．版本查看：

$ cat /usr/local/cuda/include/cudnn.h | grep CUDNN_MAJOR -A 2

得到：

#define CUDNN_MAJOR 7

#define CUDNN_MINOR 6

#define CUDNN_PATCHLEVEL 5

#define CUDNN_VERSION (CUDNN_MAJOR * 1000 + CUDNN_MINOR * 100 + CUDNN_PATCHLEVEL)

#include "driver_types.h"

3.6 Python

- 版本要求：3.7 (注：系统自带的 python2.7 不要卸载或删除，以免影响系统运行)
- 具体步骤：

1．下载：

$ wget https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tar.xz

2．编译安装：

$ tar -xvJf Python-3.7.3.tar.xz

$ cd Python-3.7.3

$ ./configure --prefix=/usr/local/bin/python3

$ sudo make

$ sudo make install

3．创建软链接：

$ ln -s /usr/local/bin/python3/bin/python3 /usr/bin/python3

$ ln -s /usr/local/bin/python3/bin/pip3 /usr/bin/pip3

4．版本查看：

$ python3 --version

3.7 Pytorch

- 版本要求：1.3.0
- 具体步骤：

  1．pip 安装：pip3 install torch==1.3.0

  注：这里的 pip3 是上面的 python3.7 下的，可通过 pip3 -V 查看

  2．版本查看：

  $ python3

  >> import torch

  >> print(torch.__version__)

3.8 Torchvision

- 版本要求：0.4.2
- 具体步骤：

  1. pip 安装：pip3 install torchvision==0.4.2

  注：这里的 pip3 是上面的 python3.7 下的，可通过 pip3 -V 查看

  2. 版本查看：

  $ python3

  >> import torchvision

  >> print(torchvision.__version__)

3.9 Magik 训练插件

- 版本要求：1.1.1
- 具体步骤：

  1. 环境确认：

  $ python

  >> import torch

  >> print(torch.__version__) #torch 的版本

  >> print(torch.version.cuda) #对应的 cuda 版本

  >> print(torch.backends.cudnn.version()) #对应的 cudnn 的版本

  也可以根据自身需要设置 cuda 和 cudnn 的版本

  2. 安装前根据上述环境在插件目录

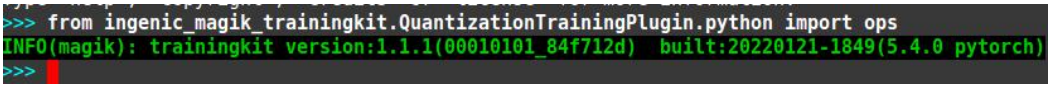
  在我们提供的插件目录下 magik-toolkit/TrainingKit/pytorch/magik_whl 下找到对应的安装包。

  pip 安装：pip3 install magik_trainingkit_torch_130-1.1.1-py3-none-any.whl

  注：这里的 whl 安装包由我们根据客户的环境提供，如果插件目录下没有对应的版本，及时同步给相关人员以适配环境编译对应的插件。另外，

若只是更新插件，需要先卸之前的旧插件。

2. 查看是否安装成功：



出现上图表明导入成功，绿色的字体中有当前使用插件的版本，commit 号及编译的日期。

3.10 其他

运行 yolov5 需要一些其他的安装包：
pandas ($ pip install pandas)
requests ($ pip install requests)
cv2 ($ pip install opencv-python)
yaml ($ pip install pyyaml)
tqdm ($ pip install tqdm)
matplotlib ($ pip install matplotlib)
seaborn ($ pip install seaborn)
运行时如还有其他，根据提示用 pip install 进行安装，直至正常运行。

4. Yolov5s 训练流程

4.1 代码和模型的下载

yolov5 的代码地址：
https://github.com/ultralytics/yolov5.git
这里因为加入了插件部分，对相关算子进行了重新的封装，我们会提供一套基于原生 yolov5 改后的训练代码 yolov5s-person, 其中原生的算子也有做注释保留，使用者以对比修改前后的算子了解和熟悉具体修改和使用方法。若对原生的代码感兴趣，或 者想先验证一下环境是否正常，也可先跑一下原生的 yolov5 的代码。
提供的代码在 magik-toolkit/Models/training/pytorch/T40/yolov5s-person/下，以下介绍的训练和转换部分均在此目录下完成。
模型的下载地址:
链接：https://pan.baidu.com/s/1LcE4-gLtGUaEplPrwt32rQ
提取码：ug3k
如图，这三个模型分别是按步骤用 coco 中的 1 万 5 的单人形（person）数据训练的 32bit，8bit 和 4bit 模型，后面的测试和上板试验用的是 yolov5s-person-4bit.pt, 下 载 放 到 magik-toolkit/Models/training/pytorch/T40/yolov5s-person/runs/train 下。

4.2 数据准备

1. 以 COCO2017 为例，提取出里面的 person(id: 0)部分并转换成 Yolo 训练需要的数据格式，COCO2017 数据集下载好并解压，annotation 文件的格式为.json 格式。
下载地址(用 wget 进行下载):
图片：
http://images.cocodataset.org/zips/train2017.zip
http://images.cocodataset.org/zips/test2017.zip
http://images.cocodataset.org/zips/val2017.zip
标注:
http://images.cocodataset.org/annotations/stuff_annotations_trainval2017.zip
http://images.cocodataset.org/annotations/image_info_test2017.zip
http://images.cocodataset.org/annotations/annotations_trainval2017.zip

2. 生成数据的脚本在 COCO_forYOLO 文件夹下：
(1)运行 python batch_split_annotation_foryolo.py (注意修改程序中的 coco 路径'coco_data_dir=')。

(2)运行完会在 coco 路径下生成 person/data/images、 person/data/ImageSets、person/data/labels 三个文件夹， 将 ImageSets 下的 train2017.txt、 val2017.txt、test2017.txt 放入 persondet/data/coco 文件夹下， txt 文件里存储图片的绝对路径。

4.3 网络训练

1. 训练配置：
默认配置参数可参考 models/commom.py 中 is_quantize, bitw, bita 等参数:

```
bita = 32

if bita==32:
    bitw = 32
    is_quantize = 0
    clip_max_value = 6.0
    shortcut_clip_max_value = 2.0
elif bita==8:
    bitw = 8
    is_quantize = 1
    clip_max_value = 6.0
    shortcut_clip_max_value = 2.0
elif bita==4:
    bitw = 4
    is_quantize = 1
    clip_max_value = 4.0
    shortcut_clip_max_value = 1.5

weight_factor = 3.0
target_device = "T40"
```

32bit, 设置: bita = 32
8bit, 设置: bita = 8
4bit, 设置: bita = 4
其中,
is_quantize – 是否进行量化, 0-不量化，1-量化，bita 为 32 时，注意该值设为 0
bitw – weight 的位宽
bita – feature 的位宽
clip_max_value – feature 的截断值，建议 8bit 设为 6.0，4bit 设为 4.0
shortut_clip_max_value 代表 shortcut 的 feature 的截断值，建议 8bit 设为 2.0，4bit 设为 1.5

2. 训练脚本 yolov5s-person/train.sh：

```
export NCCL_IB_DISABLE=1
export NCCL_DEBUG=info
GPUS=6

python3 -m torch.distributed.launch --nproc_per_node=$GPUS
--master_port=60051 train.py \
--data data/coco-person.yaml\
--cfg models/yolov5s.yaml \
--weights '' \
--batch-size 132 \
--hyp data/hyp.scratch.yaml \
--project ./runs/train/yolov5s-person-32bit \
--epochs 300 \
--device 0,1,2,3,4,5
```

(1)关于预训练
浮点训练时没有预训练模型，--weights 为 ‘ ’， 8bit 训练时加载已得到的精度足够的 32bit 模型， 4bit 加载训练好的 8bit 作为预训练模型， 这样一

步步推进， 效果更佳。

(2)关于多卡训练
训练时加入了 torch.distributed.launch，为多卡的分布式训练，GPUS 的值和下面 device 的总数对应，按实际情况对应修改，如果是单卡训练，直接用 python3 train.py 加上后面的参数即可。 batch-size 是所有显卡总的 batch 数目，按实际显卡大小设置即可。

(3)关于学习率
超参数的设置在 data/hyp.scratch.yaml 里，lr0 设置初始学习率，其余采用默认值。train.py 里加载预训练模型有关于加载 optimizer 的部分，这里因为低 bit 模型相对于 32bit 模型不是接着训练而是类似重训，所以这里建议去掉加载 optimizer 的部分，以免影响低 bit 的训练效果。

```
# Resume
start_epoch, best_fitness = 0, 0.0
if pretrained:
    # Optimizer
    #if ckpt['optimizer'] is not None:
    #    optimizer.load_state_dict(ckpt['optimizer'])
    #    best_fitness = ckpt['best_fitness']

    # EMA
    #if use_ema and ema and ckpt.get('ema'):
    #    ema.ema.load_state_dict(ckpt['model'].float().state_dict())
    #    ema.updates = ckpt['updates']

    # Epochs
    # start_epoch = ckpt['epoch'] + 1

    if resume:
        start_epoch = ckpt['epoch'] + 1
        assert start_epoch > 0, '%s training to %g epochs is finished, nothing to resume.' % (w
ights, epochs)
```

ema 和 epoch 也存在类似的问题，因此如图也对应做了修改。

(4)模型的保存
train.py 里--project 可设置保存模型路径, 在 runs/train/project 下， weights 下保存的有两个模型，best.pt 本次训练到目前最好的， last.pt 本次训练到目前最新的， 每个 epoch 测试的结果保存在 result.txt 中, 可随时查看。

(5)训练经验
针对 yolov5 系列的训练，目前总结的经验是 32bit 用 sgd 和 lr0.01，8bit 基于 32bit 的预训练用 sgd 和 lr0.01，4bit 基于 8bit 的预训练用 adam 和 lr0.001。具体命令参见 train.sh。32bit 没有预训练，收敛稍慢；8bit 有预训练且表达能力可以，收敛较快；4bit 虽有预训练但表达能力稍弱，收敛稍慢。

4.4 模型测试

1.待检测图片
通过 python3 detect.py -h 查看选取所需参数， 通过设置 source 可检测图片或视频或图片文件夹, 检测结果可设置显示(--view-img)或保存(--save-img)
- Image: `--source file.jpg`

- Video: `--source file.mp4`
- Directory: `--source dir/`

示例如下，若需要其他操作可选择相应的参数;

```
$ sh detect.sh(python detect.py --source data/images/bus.jpg \
 --weights ./runs/train/yolov5s-person-4bit.pt \
 --imgs 640 --device 0 – view-img)
```

其中，

source – 待检测图片

weights – 训练好的用于测试的模型

imgs – 测试图片的大小

device – 用第几块显卡测试

view-img – 是否画框显示检测结果

注：检测的模型配置一定要和训练时候的配置(bita)一致.

2. 测试模型精度

```
$ sh test.sh(python test.py --data data/coco-person.yaml \
 --weights ./runs/train/yolov5s-person-4bit.pt \
 --imgs 640 --device 0 --batch-size 40)
```

待测试的模型通过---weights 指定， 验证集通过 data/coco-person.yaml 中的 val2017.txt 确定，其余参数根据实际需要给定。

具体测试结果如下：

640x640

	Class	Images	Targets	P	R	mAP@0.5	mAP@.5:.95
32bit:	all	5000	11004	0.771	0.615	0.700	0.422
8bit:	all	5000	11004	0.751	0.638	0.706	0.430
4bit:	all	5000	11004	0.786	0.602	0.698	0.419

models 下的 yolov5m.yaml 和 yolov5l.yaml 也分别对应原始的 yolov5m 和 yolov5l，训练流程也都同 yolov5s，有需要可以直接使用。

5. 模型转换

5.1 pt 转 onnx

在上述训练的环境下生成 onnx 文件：

```
$ sh convert_onnx.sh(python convert_onnx.py \
--weights ./runs/train/yolov5s-person-4bit.pt)
```

input：指定待转模型的路径

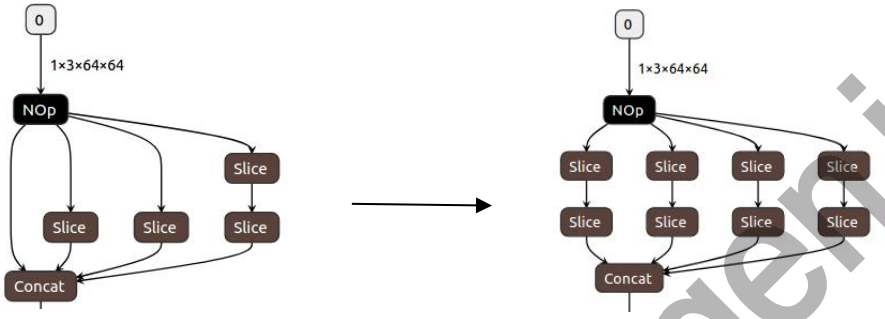
output：.onnx，位置和.pt 在同一目录

转 onnx 时必须使用 opset9, 否则会产生多余的节点导致后续转换工具不支持，如因使用 opset9 转 onnx 失败，可参见步骤 7 的疑问解答，不同版本的

torch 会有一些不同的需要注意的点。

```
def _slice(g, input, axes, starts, ends):
    assert len(starts) == len(ends)
    # if len(starts) == 1 and starts[0] == 0 and ends[0] == 9223372036854775
    #     return input
    return g.op("Slice", input, axes_i=axes, starts_i=starts, ends_i=ends)
```

特别地，对于 yolov5 用到了 focus 算子的转 onnx 之前需要注意修改 torch/onnx/symbolic_opset9.py 中的_slice()函数，如下：



改前　　改后

转换后的 onnx 文件可用 netron 进行可视化查看。

5.2 onnx 转 bin

这一步不依赖 python，torch 及插件等环境，只需要我们的转换工具 magik-transform-tools(在 magik-toolkit/TransformKit 下)及转换脚本 run.sh：

cd transform

$ sh run.sh

具体地，

path=../../../../../../TransformKit

$path/magik-transform-tools \
--framework onnx \
--target_device T40 \
--outputpath yolov5s-person-4bit.mk.h \
--inputpath ../runs/train/yolov5s-person-4bit.onnx \
--mean 0,0,0 \
--var 255,255,255 \
--img_width 416 \
--img_height 416 \
--img_channel 3

其中，

inputpath – 转换的输入，上述得到的 onnx 文件

outputpath – 转换的输出，转换完成后会保存在当前目录下生成所需的.bin 文件，这时候模型的大小才会按需求位宽减小
target_device – 和训练时的设置保持一致
mean – 训练时的预处理的均值
var – 训练时的预处理的方差
img_width – 上板运行时的输入的宽(这里用的训练时的设置，保证 32 对齐即可)
img_height – 上板运行时的输入的高(这里用的训练时的设置，保证 32 对齐即可)
img_channel – 上板运行时的输入的通道(和训练保持一致)

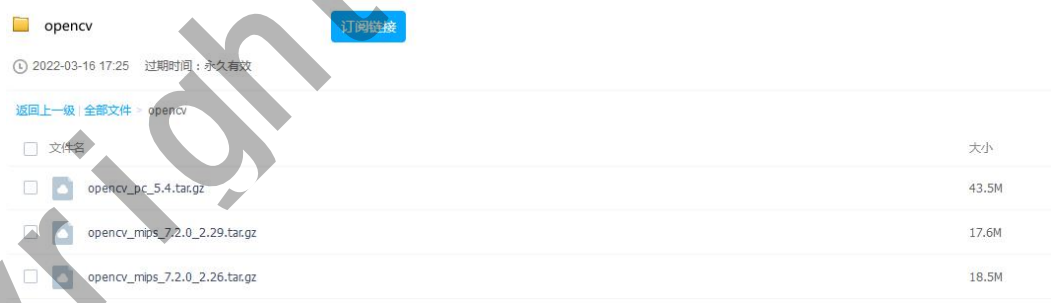
```
INFO(magik): magik-transform-tools version:1.0.0(00010000_bbb8835)  cuda version:9.0.176  built:20220214-1120_GPU
2022-02-15 14:53:37.819065: W clip_model_by_nop_optimizer.cc:48-collect_remove_nodes] Multiple output NOp exists on the current pathway!
2022-02-15 14:53:37.820572: W clip_model_by_nop_optimizer.cc:48-collect_remove_nodes] Multiple output NOp exists on the current pathway!
2022-02-15 14:53:37.820625: W clip_model_by_nop_optimizer.cc:48-collect_remove_nodes] Multiple output NOp exists on the current pathway!
2022-02-15 14:53:37.820665: W clip_model_by_nop_optimizer.cc:48-collect_remove_nodes] Multiple output NOp exists on the current pathway!
2022-02-15 14:54:00.861688: W quantize_operation.cc:305-search] Not found QuantizeOperation function for Unpool2D
2022-02-15 14:54:00.875195: W quantize_operation.cc:305-search] Not found QuantizeOperation function for Unpool2D
*********************************************************************************************
*                        ^ ^ Convert successfully, Enjoy it ^ ^                             *
*********************************************************************************************
```

转换后会在当前目录生成 yolov5s-person-4bit.bin 文件，这个就是我们最终上板要用的二进制模型文件。

6. 模型上板

6.1 代码编译

1. 准备工作：
(1) mips 编译过的 pencv 库下载地址：
链接：https://pan.baidu.com/s/1XlN4otXKUy8zgvW5G4yOwA?pwd=chab
提取码：chab



下载之后放到 magik-toolkit/ThirdParty 下进行解压:
tar -xvf opencv_mips_7.2.0_2.29.tar.gz

(2) yolov5s-person/venus_sample_yolov5s 目录下我们还提供了上板运行需要的 inference.cpp，Makfile、测试数据，另外我们还需要用到 venus 库及 mips 编译工具。其中，
venus 库在 magik-toolkit/InferenceKit/nna1 下；
mips 编译工具由方案同事提供。

2. 网络输入
这里为了用户快速实现流程，我们在这里加入了 opencv 的库函数方便调用 imread 等函数，所以测试的时候直接传入 jpg 图即可。

3. 模型的加载
提供的实例 inference.cpp 中模型是通过参数传入的，运行前注意同步拷贝到板端对应的目录并在运行时传入。

4. 超参数的设置

```
oid generateBBox(std::vector<venus::Tensor> out_res, std::vector<magik::venus::ObjBbox_t>& candidate_boxes

float person_threshold = 0.3;
int classes = 1;
float nms_threshold = 0.6;
std::vector<float> strides = {8.0, 16.0, 32.0};
int box_num = 3;
std::vector<float> anchor = {10,13,  16,30,  33,23, 30,61,  62,45,  59,119, 116,90,  156,198,  373,326};
```

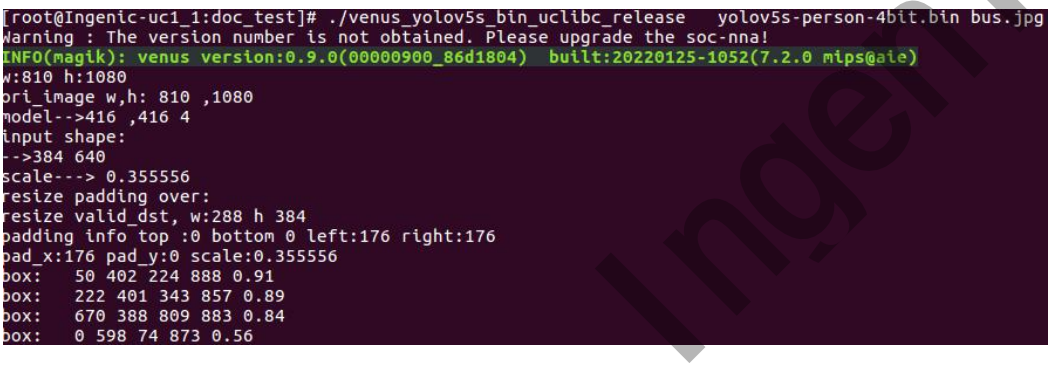
strides 和 anchor 按 yolov5 的实际使用设置，person_threshold 和 nms_threshold 分别对应原始代码中的 conf-thres（置信度阈值）和 iou-thres（iou 阈值），classes 是类别数。

5. 编译
TOPDIR – venus 库的相对目录
opencv_path – mips 编译后的 opencv 库的路径
libtype – 根据实际板子的需求确定是否 muclibc（这里以 muclibc 为例）
build_type – release 模式，运行得到结果，默认设置
– profile 模式，运行时网络结构可视化及网络每层运行时间及 GOPs 统计
– debug 模式，运行得到结果的同时保存每层量化 feature 的结果
– nmem 模式，统计模型运行时 nmem 内存占用情况，运行程序，内存使用情况保存在/tmp/nmem_memory.txt
直接 make 编译 inference.cpp 即可生成 venus_yolov5s_bin_uclibc_*，即我们上板需要的可执行文件。
注意：我们同时提供了用于实际板端运行的输入为 nv12 的代码用例 inference_nv12.cpp 及输入数据 10_w1024_h714.nv12，如有需要，可修改 Makefile 进行编译及使用测试。

6.2 上板运行

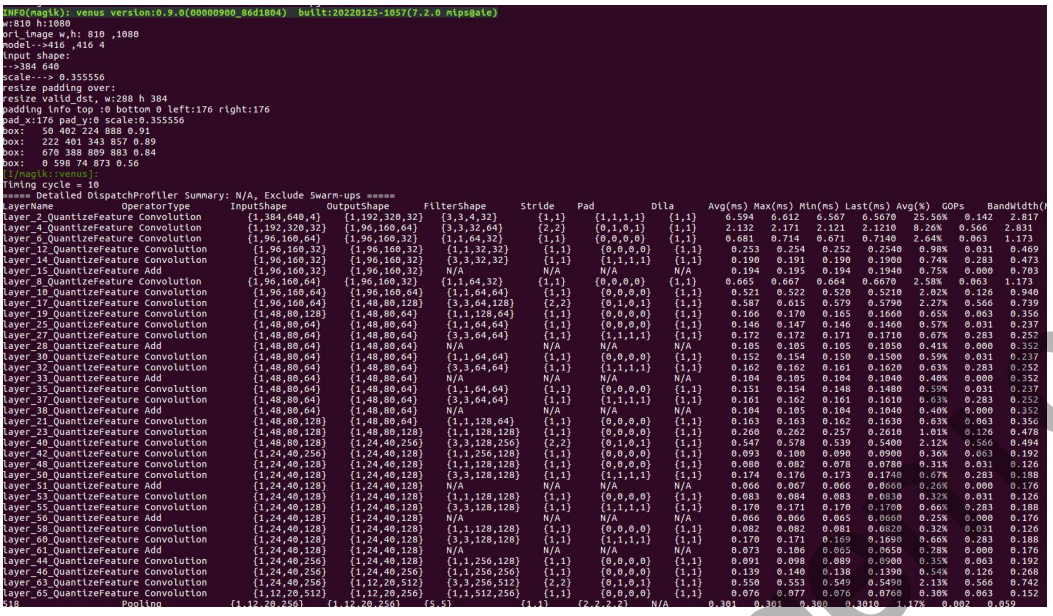
注：
opencv 库在
magik-toolkit/ThirdParty/opencv_mips_7.2.0_2.29/uclibc/lib 下
venus 库在
InferenceKit/nna1/mips720-glibc229/lib/uclibc/下

1. release(发布库)
编译: make build_type=release
在当前文件夹下生成 venus_yolov5s_bin_uclibc_release 可执行文件, 拷贝 venus 库(libvenus.so)、opencv 库(libopencv_core.so.3.3, libopencv_highgui.so.3.3,libopencv_imgcodecs.so.3.3,libopencv_imgproc.so.3.3, libopencv_videoio.so.3.3)、可执行文件(venus_yolov5s_bin_uclibc_release)、模型文件(yolov5s-person-4bit.bin)、测试图片(bus.jpg)至开发板运行即可：./venus_yolov5s_bin_uclibc_release yolov5s-person-4bit.bin bus.jpg
(注：运行前添加库路径至 LD_LIBRARY_PATH:
export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH).
清除 make build_type=release clean



2. debug（用于核对数据的库）
详见步骤 7.2，输入的处理有些不同。

3. profile（网络可视化及每层运行时间统计）
编译: make build_type=profile
在当前文件夹下生成 venus_yolov5s_bin_uclibc_profile 可执行文件，拷贝 venus 库(libvenus.p.so)、opencv 库(libopencv_core.so.3.3, libopencv_highgui.so.3.3,libopencv_imgcodecs.so.3.3,libopencv_imgproc.so.3.3, libopencv_videoio.so.3.3)、可执行文件(venus_yolov5s_bin_uclibc_profile)、模型文件(yolov5s-person-4bit.bin)、测试图片(bus.jpg)至开发板运行即可：./venus_yolov5s_bin_uclibc_profile yolov5s-person-4bit.bin bus.jpg)
注：运行前添加库路径至 LD_LIBRARY_PATH:
export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH
清除 make build_type=profile clean



4. nmem 模式（用于统计网络运行时 nmem 占用情况，保存在/tmp/nmem_memory.txt）

编译: make build_type=nmem

在当前文件夹下生成 venus_yolov5s_bin_uclibc_nmem 可执行文件，拷贝 venus 库(libvenus.m.so)、opencv 库(libopencv_core.so.3.3 ,libopencv_highgui.so.3.3,libopencv_imgcodecs.so.3.3,libopencv_imgproc.so.3.3,libopencv_videoio.so.3.3)、可执行文件(venus_yolov5s_bin_uclibc_nmem)、模型文件(yolov5s-person-4bit.bin)、测试图片(bus.jpg)至开发板运行即可：./venus_yolov5s_bin_uclibc_nmem yolov5s-person-4bit.bin bus.jpg

注：运行前添加库路径至 LD_LIBRARY_PATH:

export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH

清除 make build_type=nmem clean

```
[root@Ingenic-uc1_1:doc_test]# ./venus_yolov5s_bin_uclibc_nmem  yolov5s-person-4bit.bin bus.jpg
Warning : The version number is not obtained. Please upgrade the soc-nna!
[INFO(magik): venus version:0.9.0(00000900_86d1804)  built:20220125-1045(7.2.0 mips@aie)
w:810 h:1080
ori_image w,h: 810 ,1080
model-->416 ,416 4
input shape:
-->384 640
scale---> 0.355556
resize padding over:
resize valid_dst, w:288 h 384
padding info top :0 bottom 0 left:176 right:176
pad_x:176 pad_y:0 scale:0.355556
box:  50 402 224 888 0.91
box:  222 401 343 857 0.89
box:  670 388 809 883 0.84
box:  0 598 74 873 0.56
[root@Ingenic-uc1_1:doc_test]# cat /tmp/nmem_memory.txt
nmem total = 131072K
nmem used  = 13024K
nmem free  = 118048K
#### %9.94 ####
```

7. 数据核对

验证 PC 端和板端的数据是否能对齐，可按如下操作：

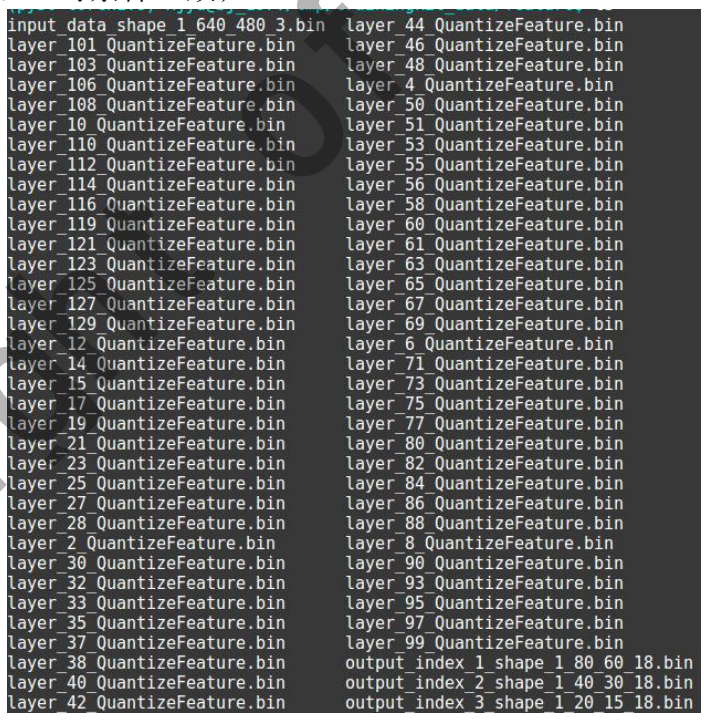
1. PC 端

步骤 4.4-1 测试单张图片加入环境变量 MAGIK_TRAININGKIT_DUMP=1 可保存每层量化结果到/tmp/trainingkit_data/feature 下，也可通过设置环境变量 MAGIK_TRAININGKIT_PATH 指定保存的目录，具体运行命令为:

```
$ MAGIK_TRAININGKIT_DUMP=1
  MAGIK_TRAININGKIT_PATH=”./” python detect.py \
    --source data/images/bus.jpg \
    --weights ./runs/train/yolov5s-person-4bit.pt \
    --imgs 640 --device 0
```

这里的 bus.jpg 原始分辨率是 1080*810, 测试时加入了目标 imgs 为 640，按 yolov5 代码的缩放原则（长边 640，短边等比缩放再填充至 32 的倍数），最后进入测试的分辨率为 640*480，最终保存./trainingkit_data/feature 下（具体见下图），可见输入层有保存为 input_data_shape_1_640_480_3.bin，后面的 shape 命名的规则是 n,h,w,c, 即高为 640，宽为 480，最后面是三个输出层；

参数 imgs 、conf-thres、iou-thres 的设置也是为了和 c 代码(详见 6.1-3)保持一致，以保证比对条件一致；



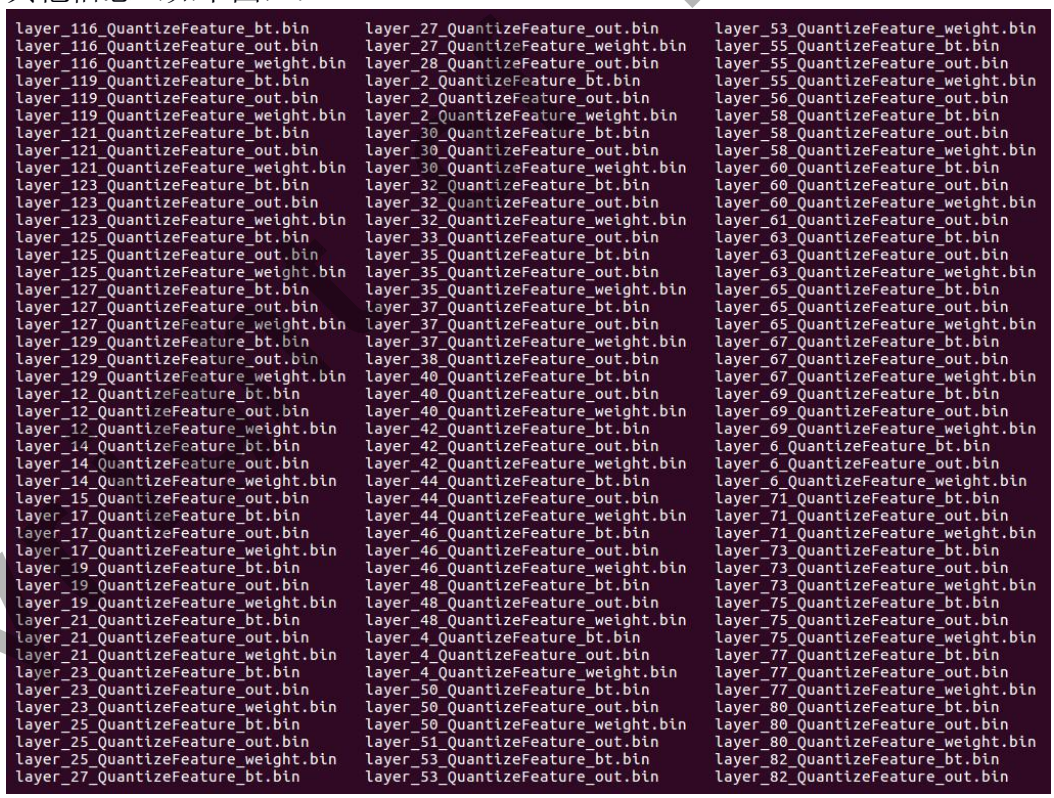
2. 板端

(1)板端需要的输入数据为 4 通道，对于上面 pc 端测试的 bus.jpg，我们用运行时保存的输入 input_data_shape_1_640_480_3.bin 处理成.h 之后拿来做板端的测试，以保证二者的输入完全一致，可先做线下处理(脚本在

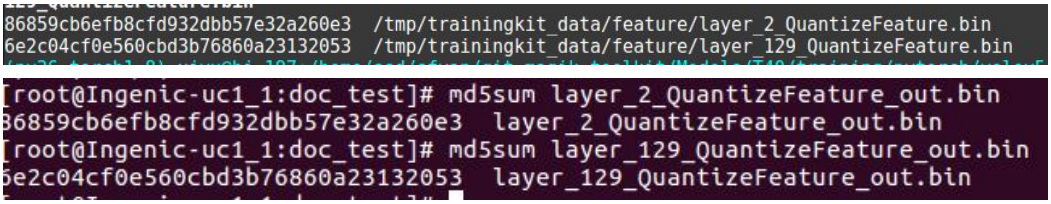
yolov5s-person/venus_sample_yolov5s/generate_img_input.py):
python generate_img_input.py bin_path w h c bin
bin_path -- 要转换的 bin 文件，即 pc 端运行保存的输入 bin
w, h -- bin 文件对应的宽和高，bin 文件名字里有对应, 分别是 1_h_w_c
bin -- flag 标志，表示由 bin 文件转头文件，还有一种是 img 转头文件
说明：用 bin 的话可以完全保证两边的输入一致，方便对数据。

(2)提供的实例中模型是通过参数传入的(argv[1])，运行时注意同步拷贝到板端对应的目录，运行时按需传入。宽(w)、高(h)也要从外部传入，具体值同(1)中。

(3)编译 c 代码：编译之前最好先将之前编译过的模式做一下 clean(make build_type=release/profile/nmem clean), 不然可能出现 opencv 的链接库失败的错误，之后 make build_type=debug，在当前文件夹下生成 venus_yolov5s_bin_uclibc_debug 可执行文件，拷贝 venus 库(libvenus.d.so)、可执行文件(venus_yolov5s_bin_uclibc_debug)、模型文件(yolov5s-person-4bit.bin)至开发板运行：
./venus_yolov5s_bin_uclibc_debug yolov5s-person-4bit.bin 480 640
其中，480 对应为 w，640 对应 h，运行的时候就会自动保存每层的输出及其他信息（如下图）：



其中 layer_*_QuantizeFeature_out.bin 和 PC 端运行时保存在./trainingkit_data/feature 下的 layer_*_QuantizeFeature.bin
是一一对应的，直接核对 md5 值是否一致即可，在保证输入完全一致的情况下中间有层不对应及时反馈。



debug 模式这里只核对网络结果，没有加后处理部分，因此无结果打印。

8. 常见疑问解答

1. 问：为什么 release 运行 810*1080 的 bus.jpg 的分辨率是 640*384，debug 的时候是 480*640？
答：当我们设置目标分辨率为 640 时，板端的缩放规则是按等比缩放之后 w 填充至 640（因为实际的视频流大都是 w>h），这里 640 是 w，384 是 h；而 yolov5 代码里的规则是按长边 640 等比缩放再将短边填充至 32 的倍数，所以这里 h 是 640,480 是 w，debug 是和 pc 端保持一致的，实际使用按需设置即可。

2. 问：导入 magik 包出现“undefined symbol: _ZN6caffe28TypeMeta21_typeMetaDataInstanceIN3c108BFloat16EEEPKNS_6detail12TypeMetaDataEv”类似错误？
答：当前环境下 torch 的版本和 magik 包的不对应。

3. 问：“importError: version ‘libcudart.so.10.1’ not found”？
答：当前 cuda 版本和插件编译时的版本不对应 。

4. 问：加插件之后的网络是否可以加载原生 torch 模型做预训练？
答：理论可以，层对应好就行，不过基于原生 float 的 32bit 模型还是要再微调训练一下，不建议直接加载原生的跑 8bit。

5. 问：训练出现“RuntimeError: Address already in use”？
答：在 python -m torch.distributed.launch 后指定一个未被使用的端口—master_port=60053

6. 问：训练出现“expected scalar type Float but found Half”？
答：torch 中用了 amp.autocast 进行半精度训练加速,但量化暂不支持半精度训练，注掉这块即可。另，原生 v5 在保存模型时按 half()存的，目前发现量化这样操作最后存的模型的精度会有损失，建议去掉。

7. 问：torch 转 onnx 出现“step!=1 is currently not supported”？
答：yolov5 slice 转换遇到的问题, 1.6 及之前版本可以可以,1.7 以后变成 error 了，torch/onnx/symbolic_opset9.py 对应报错的行 if + raise RuntimeError 注掉即可。

8. 问：torch 转 onnx 出错“RuntimeError:input_shape_value== reshape_value || input_shape_value==1 || reshape_value == 1 INTERNAL ASSERT FAILED ……”？
答：1.9 版本之后的错，torch/onnx/utils.py 中 635 行_export()最后一个参数的 onnx_shape_inference=True 改为 onnx_shape_inference=False；
torch1.10 是在 677 行。不同版本找这个函数相应修改即可。

9. 问：torch1.8/torch1.9 版本转换 onnx 会出现“Warning: Unsupported operator NOp. No schema registered for this operator.”等？
答：warning 不用管,不影响结果。

10. 问：转 bin 出现“After checking the input dimension, the input of concat cannot be concatenated!”？
答：slice 的问题（参见 5.1），将 torch/onnx/symbolic_opset9.py 中_slice 函数中的 if+return 这两句去掉，可视化 onnx 的 slice 正常。

11. 问：c 代码编译出现“error: 'constexpr bool std::isinf(double)' conflicts with a previous declaration”等一系列冲突错误？
答：mips 的编译器和 x86 自带的出现冲突，将环境变量 CPLUS_INCLUDE_PATH 中关于/usr/include/x86_64-linux-gnu 的部分去掉。

12. 问：c 端编译出现“make: mips-linux-gnu-g++: Command not found”？
答：未指定 mips 编译器，通过 export 设置。

13. 问：板端精度和 PC 端精度损失有多少？
答：理论上损失很小，1 个点以内，若出现较大差距，可检查输入是否一致，训练和验证的数据是 BGR/RGB，阈值的设置，也可对比网络输出结果看看输出是否一致，具体见步骤 7.

14. 问：转 onnx 的分辨率，转换工具的分辨率以及板端测试的分辨率需要保持一致吗？
答：最好保持一致，尤其后两者，本例中没有保持一致是因为对这个没有影响，对于其他非全卷积网络可能导致转换出来的模型有影响，所以测试验证集精度时建议输入数据统一线下处理到同一大小再做 PC 端和板端的比较。