

基于1684芯片下SOC模式跑通Demo

基于1684芯片下SOC模式跑通Demo

概要

基本概念

环境准备

跑通Demo

1. 下载sophon-demo包

2. 准备模型和数据

3. 模型编译

tpu-nntc环境搭建

生成FP32 Bmodel

生成INT8 Bmodel

bmrt_test(可选)

4. SOC例程 (C++)

环境准备

程序编译

推理测试

5. SOC例程 (python)

环境准备

推理测试

常见问题

文档链接

概要

本手册适用于算丰基于[BM1684芯片](#)的产品：[SE5系列](#)，[SE6系列](#)

基本概念

[算能SDK入门文档](#)

模型迁移：将原始深度学习框架（pytorch, caffe, onnx, paddle等）下训练生成的模型转换为BM1684平台运行的FP32 BModel，必要时使用BMLang开发不支持的算子；使用量化集量化生成INT8 BModel，并测试调优确保INT8 BModel精度符合要求（推荐使用4N Batch BModel以获得最佳性能）。

算法移植：基于BM1684硬件加速接口，对模型的前后处理及推理现有算法进行移植。

程序移植：移植任务管理、资源调度等算法引擎代码及逻辑处理、结果展示、数据推送等业务代码。

测试调优：网络性能与精度测试、压力测试，基于网络编译、量化工具、多卡多芯、任务流水线等方面的深度优化。

部署联调：将算法服务打包部署到BM1684硬件产品上，并在实际场景中与业务平台或集成平台进行功能联调；必要时在生产环境中调整参数配置并收集数据进一步优化模型。

环境准备

下载官方SDK：[下载链接V23.03.01](#)当前推荐版本：建议使用的SDK版本为V23.03.01

产品文档

开发资料

课程中心

SDK

Linux

V23.05.01

V23.03.01

V22.12.01

V22.11.01

V22.10.01

V22.09.02

V3.0.0

V2.7.0

daily build

cv183x_cv182x_arm_gcc

riscv_gcc_v2.6.1

Others

Modules

v23.03.01 SDK-23.03.01

23.03.01版本SDK
压缩包内的文件清单含有各模块简要说明

v23.03.01 SophonSDK 开发指南

v23.03.01 LIBSOPHON 使用手册

v23.03.01 Sophon-Sail 用户手册

v23.03.01 BMLIB 开发参考手册

v23.03.01 BMRuntime 开发参考手册

2023-04-03

2023-04-19

2023-04-19

2023-04-19

2023-04-19

2023-04-19

下载

【SOC模式环境安装】

1. 在解压后的SDK中，找到sophon-img文件夹下的sdcard刷机包。（参考SE5固件升级文件）
2. 将设备的LAN口与电脑LAN口直连，并配置电脑IP到192.168.150.x网段，通过ssh linaro@192.168.150.1连接到设备，默认密码也是linaro
3. 将盒子WAN口连入局域网，并按需修改静态IP，后续操作可以通过WAN口进行。

请注意，对于SoC平台，安装好SophonSDK(>=v22.09.02)后内部已经集成了相应的libsophon、sophon-opencv和sophon-ffmpeg运行库包，位于/opt/sophon/下，可直接用于运行环境。通常在x86主机上交叉编译程序，使之能够在SoC平台运行。

安装成功后在盒子上或者在插板卡的服务器中使用bm-smi命令，能运行即可。
请注意刷完机后SDK Version应该是0.4.6

跑通Demo















可以[根据GitHub上的步骤跑通](#)，也可参考如下步骤：

1. 下载sophon-demo包

推荐方式一：（clone的为最新版本，方式二是稳定版本）

```
git clone "https://github.com/sophgo/sophon-demo.git"
```

推荐方式二：在SDK里找到sophon-demo

名称	修改日期	大小	种类
>  bmmonkey_20230327_070017	2023年7月30日 02:03	--	文件夹
>  bmpanda_20230327_070053	2023年7月30日 02:03	--	文件夹
>  libsophon_20230327_025400	2023年7月30日 02:04	--	文件夹
>  sophon-demo_20230327_085900	2023年7月30日 01:59	--	文件夹
>  sophon-img_20230327_063808	2023年7月30日 02:04	--	文件夹
>  sophon-mw_20230327_040051	2023年7月30日 02:03	--	文件夹
>  sophon-pipeline_20230327_081409	2023年7月30日 02:03	--	文件夹
>  sophon-rpc_20230327_065739	2023年7月30日 02:03	--	文件夹
>  sophon-sail_20230327_085400	前天 15:13	--	文件夹
>  sophon-testhub_20221013_211859	2023年7月30日 02:00	--	文件夹
>  tpu-kernel_20230327_065210	2023年7月30日 02:03	--	文件夹
>  tpu-mlir_20230327_063052	2023年7月30日 02:13	--	文件夹
>  tpu-nntc_20230327_061123	2023年7月30日 02:04	--	文件夹
>  tpu-perf_v1.1.10	2023年7月30日 02:03	--	文件夹

2. 准备模型和数据

解压并进入sophon-demo文件

```
安装unzip, 若已安装请跳过, 非ubuntu系统视情况使用yum或其他方式安装
sudo apt install unzip
chmod -R +x scripts/
./scripts/download.sh
```

该脚本下载的模型和数据如下：

```
./models
├── BM1684
│   ├── yolov5s_v6.1_3output_fp32_1b.bmodel # 使用TPU-NNTC编译, 用于BM1684的FP32 BModel, batch_size=1
│   ├── yolov5s_v6.1_3output_int8_1b.bmodel # 使用TPU-NNTC编译, 用于BM1684的INT8 BModel, batch_size=1
│   └── yolov5s_v6.1_3output_int8_4b.bmodel # 使用TPU-NNTC编译, 用于BM1684的INT8 BModel, batch_size=4
├── BM1684X
│   ├── yolov5s_v6.1_3output_fp32_1b.bmodel # 使用TPU-MLIR编译, 用于BM1684X的FP32 BModel, batch_size=1
│   ├── yolov5s_v6.1_3output_fp16_1b.bmodel # 使用TPU-MLIR编译, 用于BM1684X的FP16 BModel, batch_size=1
│   ├── yolov5s_v6.1_3output_int8_1b.bmodel # 使用TPU-MLIR编译, 用于BM1684X的INT8 BModel, batch_size=1
│   └── yolov5s_v6.1_3output_int8_4b.bmodel # 使用TPU-MLIR编译, 用于BM1684X的INT8 BModel, batch_size=4
├── BM1684_ext # 相关单输出模型, 此处没有benchmark, 用户自行使用。
├── BM1684X_ext # 相关单输出模型, 此处没有benchmark, 用户自行使用。
├── torch
│   ├── yolov5m_v6.1_1output_torchscript.pt # 相关单输出模型, 此处没有benchmark, 用户自行使用。
│   └── yolov5s_v6.1_3output.torchscript.pt # trace后的torchscript模型
├── onnx
│   ├── yolov5m_v6.1_1output_1b.onnx # 相关单输出模型, 此处没有benchmark, 用户自行使用。
│   ├── yolov5m_v6.1_1output_4b.onnx # 相关单输出模型, 此处没有benchmark, 用户自行使用。
│   └── yolov5s_v6.1_3output.onnx # 导出的onnx动态模型

./datasets
├── test # 测试图片
├── test_car_person_1080P.mp4 # 测试视频
├── coco.names # coco类别名文件
├── coco128 # coco128数据集, 用于模型量化
├── coco
│   ├── val2017_1000 # coco val2017_1000数据集: coco val2017中随机抽取的1000张样本
│   └── instances_val2017_1000.json # coco val2017_1000数据集标签文件, 用于计算精度评价指标
```

3. 模型编译

模型编译是将前端框架的模型，例如pytorch, onnx, caffe等转化为能在算能硬件跑的bmodel格式，这一步为离线编译。可以参考上方的模型图片中后缀为bmodel的文件，即为我们模型编译后的文件。

目前，针对于1684芯片，我们使用的工具链为tpu-nntc，位置也在sdk中：

tpu-nntc环境搭建

1. 安装Docker

如果已经安装docker，请跳过。

```
# 安装docker
sudo apt-get install docker.io
# docker命令免root权限执行
# 创建docker用户组，若已有docker组会报错，没关系可忽略
sudo groupadd docker
# 将当前用户加入docker组
sudo usermod -aG docker $USER
# 切换当前会话到新group或重新登录重启x会话
newgrp docker
```

提示：需要logout系统然后重新登录，再使用docker就不需要sudo了。

2. 下载并解压TPU- NNTC

从之前下载好的V23.03.01SDK包中提取出tpu-nntc_XXXX.tar.gz，将它拷贝到一台x86主机上，并创建目录然后解压。

```
mkdir tpu-nntc
# 将压缩包解压到tpu-nntc
tar zxvf tpu-nntc_vx.y.z-<hash>-<date>.tar.gz --strip-components=1 -C tpu-nntc
```

3. 创建并进入docker

TPU-NNTC使用的docker是sophgo/tpuc_dev:2.1, docker镜像和tpu-nntc有绑定关系，少数情况下有可能更新了tpu-nntc，需要新的镜像。

```
cd tpu-nntc
# 进入docker，如果当前系统没有对应镜像，会自动从docker hub上下载
# 这里将tpu-nntc的上一级目录映射到docker内的/workspace目录,用户需要根据实际情况将demo的目录映射到docker里面
# 这里用了8001到8001端口映射，之后在使用ufw可视化工具会用到
# 如果端口已经占用，请更换其他未占用端口，后面根据需要更换进行调整
docker run --name myname -v $PWD/../../workspace -p 8001:8001 -it sophgo/tpuc_dev:v2.1
# 此时已经进入docker，并在/workspace目录下
# 下面初始化软件环境
cd /workspace/tpu-nntc
source scripts/envsetup.sh
```

请注意，每次进入docker都需要重新source环境。
(备选，上述完成后可跳过) docker镜像的三种下载方式介绍：

1. [官网下载](#)，然后docker load -i xxx.docker（适用于离线）

SDK	开发镜像	
Modules		
Docker		
开发镜像	v23.03.01 工具链 docker 镜像 for MLIR 工具链 docker 镜像 for MLIR，配套 SDK-23.03.01及以后的版本	2023-04-03 下载
Device Plugin	v23.03.01 工具链 docker 镜像 for NNTC	2023-04-03
Prometheus Exporter		
	v22.09.02 工具链 docker 镜像	2022-10-24
	v3.0.0 Ubuntu18.04开发镜像 for sdk 3.0.0	2022-07-19
	v2.7.0 Ubuntu16.04开发镜像 for sdk 2.7.0	2021-11-08
	v2.6.0 Ubuntu16.04开发镜像 for sdk 2.6.0及以下	2022-01-05
	Ubuntu18.04开发镜像 for CVITEK	2022-10-26

2.

```
# 直接下载docker
docker pull sophon/tpuc_dev:v2.1
```

3.

```
# 下载并进入docker，包含docker run命令
source scripts/docker_setup.sh ..
```

生成FP32 Bmodel

在~/sophon-demo/sample/YOLOv5目录下运行

```
./scripts/gen_fp32bmodel_nntc.sh BM1684
```

生成INT8 Bmodel

在~/sophon-demo/sample/YOLOv5目录下运行

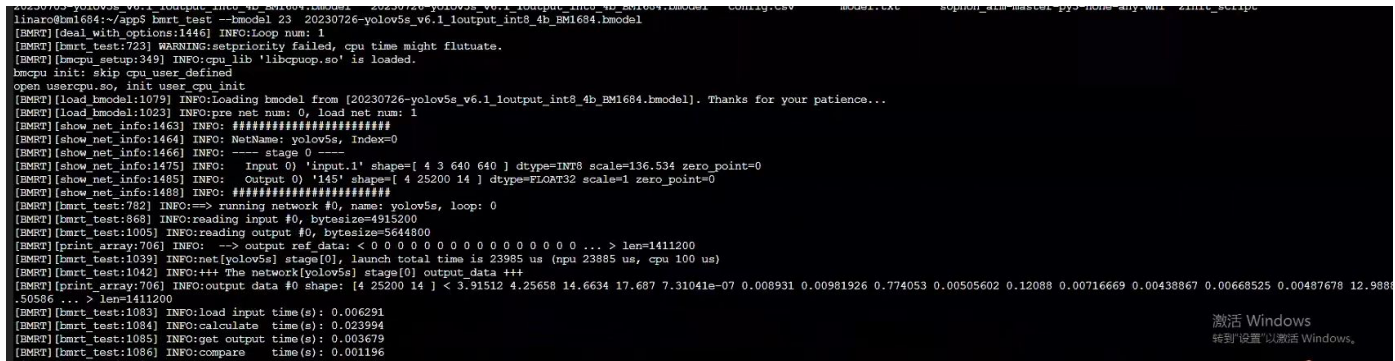
```
./scripts/gen_int8bmodel_nntc.sh BM1684
```

请注意，以上的脚本中的model来自于download.sh脚本中下载的模型。所以如果找不到模型请注意是否运行了该脚本；其次检查脚本中模型路径是否一致。

bmrt_test(可选)

上述脚本会在models/BM1684下生成对应的yolov5s_v6.1_3output_int8_1b.bmodel等文件，pcie模型可以直接使用下面命令，SOC模式将模型拷贝到设备上运行下面命令

```
bmrt_test --bmodel xxxx.bmodel
```



在输出的最后可以看到推理一次模型的推理时间（calculate time），这里的数据为模拟数据。

4. SOC例程（C++）

环境准备

如果您使用SoC平台（如SE、SM系列边缘设备），刷机后在/opt/sophon/下已经预装了相应的libsophon、sophon-opencv和sophon-ffmpeg运行库包，可直接使用它作为运行环境。通常还需要一台x86主机作为开发环境，用于交叉编译C++程序。

交叉编译环境搭建

需要在x86主机上使用SOPHON SDK搭建交叉编译环境，将程序所依赖的头文件和库文件打包至soc-sdk目录中。
A. 安装交叉编译工具链

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

如果报错：/lib/aarch64-linux-gnu/libc.so.6: version 'GLIBC_2.33' not found。这是由于您主机上的交叉编译工具链版本太高导致，可以通过如下命令重新安装：

```
sudo apt remove cpp-*-aarch64-linux-gnu
sudo apt-get install gcc-7-aarch64-linux-gnu g++-7-aarch64-linux-gnu
sudo ln -s /usr/bin/aarch64-linux-gnu-gcc-7 /usr/bin/aarch64-linux-gnu-gcc
sudo ln -s /usr/bin/aarch64-linux-gnu-g++-7 /usr/bin/aarch64-linux-gnu-g++
```

B.打包libsophon

从算能官网下载符合环境依赖的sophon-img安装包，其中包括libsophon_soc_x.y.z_aarch64.tar.gz，x.y.z表示版本号，并进行解压。

```
# 创建依赖文件的根目录
mkdir -p soc-sdk
# 解压libsophon_soc_x.y.z_aarch64.tar.gz
tar -zxf libsophon_soc_${x.y.z}_aarch64.tar.gz
# 将相关的库目录和头文件目录拷贝到依赖文件根目录下
cp -rf libsophon_soc_${x.y.z}_aarch64/opt/sophon/libsophon-${x.y.z}/lib ${soc-sdk}
cp -rf libsophon_soc_${x.y.z}_aarch64/opt/sophon/libsophon-${x.y.z}/include ${soc-sdk}
```

C.打包sophon-ffmpeg和sophon-opencv

从算能官网下载符合环境依赖的sophon-mw安装包，其中包括sophon-mw-soc_x.y.z_aarch64.tar.gz，x.y.z表示版本号，并进行解压。

```
# 解压sophon-mw-soc_x.y.z_aarch64.tar.gz
tar -zxf sophon-mw-soc_${x.y.z}_aarch64.tar.gz
# 将ffmpeg和opencv的库目录和头文件目录拷贝到soc-sdk目录下
cp -rf sophon-mw-soc_${x.y.z}_aarch64/opt/sophon/sophon-ffmpeg_${x.y.z}/lib ${soc-sdk}
cp -rf sophon-mw-soc_${x.y.z}_aarch64/opt/sophon/sophon-ffmpeg_${x.y.z}/include ${soc-sdk}
cp -rf sophon-mw-soc_${x.y.z}_aarch64/opt/sophon/sophon-opencv_${x.y.z}/lib ${soc-sdk}
cp -rf sophon-mw-soc_${x.y.z}_aarch64/opt/sophon/sophon-opencv_${x.y.z}/include ${soc-sdk}
```

这里，交叉编译环境已经搭建完成，接下来可以使用打包好的soc-sdk编译需要在SoC平台上运行的程序。

程序编译

交叉编译环境搭建好后，使用交叉编译工具链编译生成可执行文件：

1.使用bmcv接口方式

```
cd cpp/yolov5_bmcv
mkdir build && cd build
#请根据实际情况修改-DSDK的路径，需使用绝对路径。
cmake -DTARGET_ARCH=soc -DSDK=/path_to_sdk/soc-sdk ..
make
```

编译完成后，会在yolov5_bmcv目录下生成yolov5_bmcv.soc。

2.使用sail接口方式

如果您使用sophon-sail接口，需要参考交叉编译安装sophon-sail，参考[soc环境配置sophon-sail](#)然后进行如下步骤。

```
cd cpp/yolov5_sail
mkdir build && cd build
#请根据实际情况修改-DSDK和-DSAIL_PATH的路径，需使用绝对路径。
cmake -DTARGET_ARCH=soc -DSDK=/path_to_sdk/soc-sdk -DSAIL_PATH=/path_to_sail/sophon-sail/build_soc/sophon-sail ..
make
```


编译完成后，会在yolov5_sail目录下生成yolov5_sail.soc。

推理测试

A. 参数说明

可执行程序默认有一套参数，请注意根据实际情况进行传参，yolov5_bmcv.pcie与yolov5_sail.pcie参数相同。以yolov5_bmcv.pcie为例，具体参数说明如下：（这里soc和pcie的参数是一样的，只是要注意命令后缀即可）

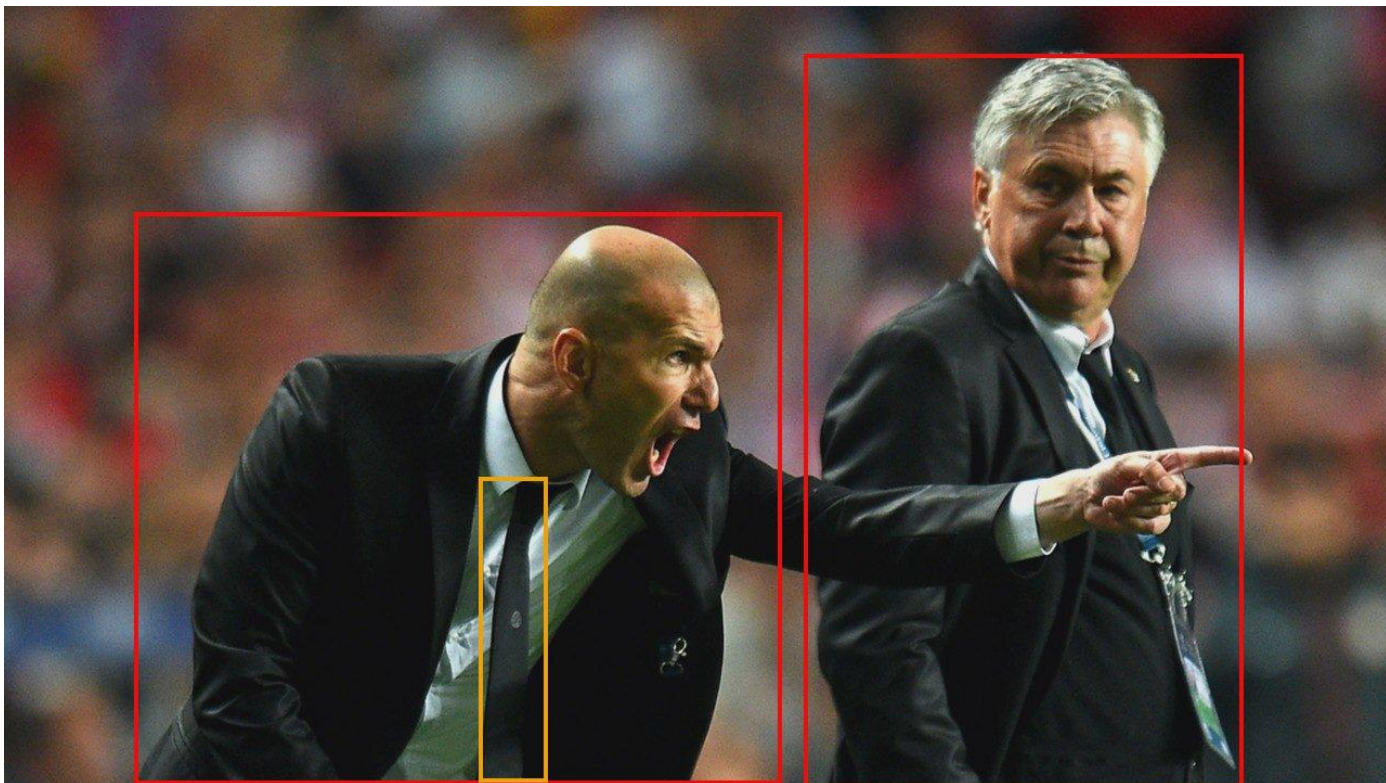
```
Usage: yolov5_bmcv.pcie [params]
      --bmodel (value:../../models/BM1684/yolov5s_v6.1_3output_fp32_1b.bmodel)
            bmodel file path
      --classnames (value:../../datasets/coco.names)
            class names file path
      --conf_thresh (value:0.5)
            confidence threshold for filter boxes
      --dev_id (value:0)
            TPU device id
      --help (value:true)
            print help information.
      --input (value:../../datasets/test)
            input path, images direction or video file path
      --nms_thresh (value:0.5)
            iou threshold for nms
```

B. 测试图片

图片测试示例如下，支持对整个图片文件夹进行测试。

```
./yolov5_bmcv.pcie --input=../../datasets/test --
bmodel=../../models/BM1684/yolov5s_v6.1_3output_fp32_1b.bmodel --dev_id=0 --
conf_thresh=0.5 --nms_thresh=0.5 --classnames=../../datasets/coco.names
```


测试结束后，会将预测的图片保存在results/images下，预测的结果保存在results/yolov5s_v6.1_3output_fp32_1b.bmodel_test_bmcv_cpp_result.json下，同时会打印预测结果、推理时间等信息。



C. 测试视频

视频测试实例如下，支持对视频流进行测试。

```
./yolov5_bmcv.pcie --input=../../datasets/test_car_person_1080P.mp4 --  
bmodel=../../models/BM1684/yolov5s_v6.1_3output_fp32_1b.bmodel --dev_id=0 --  
conf_thresh=0.5 --nms_thresh=0.5 --classnames=../../datasets/coco.names
```

测试结束后，会将预测结果画在图片上并保存在results/images中，同时会打印预测结果、推理时间等信息。

5. SOC例程（python）

环境准备

如果您使用SoC平台（如SE、SM系列边缘设备），并使用它测试本例程，刷机后在/opt/sophon/下已经预装了相应的libsophon、sophon-opencv和sophon-ffmpeg运行库包。您还需要交叉编译安装sophon-sail，具体可参考[交叉编译安装sophon-sail](#)。

此外您可能还需要安装其他第三方库：

```
pip3 install 'opencv-python-headless<4.3'
```

python例程不需要编译，可以直接运行，PCIe平台和SoC平台的测试参数和运行方式是相同的。

推理测试

A. 参数说明

yolov5_opencv.py和yolov5_bmcv.py的参数一致，以yolov5_opencv.py为例：

```
usage: yolov5_opencv.py [--input INPUT_PATH] [--bmodel BMODEL] [--dev_id DEV_ID]
                        [--conf_thresh CONF_THRESH] [--nms_thresh NMS_THRESH]
--input: 测试数据路径，可输入整个图片文件夹的路径或者视频路径；
--bmodel: 用于推理的bmodel路径，默认使用stage 0的网络进行推理；
--dev_id: 用于推理的tpu设备id；
--conf_thresh: 置信度阈值；
--nms_thresh: nms阈值。
```

B. 测试图片

图片测试实例如下，支持对整个图片文件夹进行测试。

```
python3 python/yolov5_opencv.py --input datasets/test --bmodel
models/BM1684/yolov5s_v6.1_3output_fp32_1b.bmodel --dev_id 0 --conf_thresh 0.5 --
nms_thresh 0.5
```

测试结束后，会将预测的图片保存在results/images下，预测的结果保存在results/yolov5s_v6.1_3output_fp32_1b.bmodel_test_opencv_python_result.json下，同时会打印预测结果、推理时间等信息。



C. 测试视频

视频测试实例如下，支持对视频流进行测试。

```
python3 python/yolov5_opencv.py --input datasets/test_car_person_1080P.mp4 --bmodel  
models/BM1684/yolov5s_v6.1_3output_fp32_1b.bmodel --dev_id 0 --conf_thresh 0.5 --  
nms_thresh 0.5
```

测试结束后，会将预测的结果画在results/test_car_person_1080P.avi中，同时会打印预测结果、推理时间等信息。

yolov5_bmcv.py会将预测结果画在图片上并保存在results/images中。

常见问题

文档链接
