1、 生成 trace 的 torch 模型（此操作需要再 yolox 源码中进行操作）

python3 tools/export_trace.py --output-name models/yolox_s.pt -n yolox-s -c models/yolox_s.pth

2、 fp32bmodel 转换

python3 -m bmnetp --net_name=yolox_s --target=BM1684 --opt=1 --cmp=true --shapes="[1,3,640,640]" --model=/workspace/YOLOX/models/yolox_s.pt --outdir=/workspace/YOLOX/models/yolox_s_fp32_batch1 --dyn=false

3、 int8bmodel 转换

① 、创建 lmdb 数据集文件列表

使用 create_imagelist.py 创建文件列表文件，IMG_DIR 表示图片文件夹路径，max_image_count 表示创建最多使用的额图片数量，转换时并不需要太多的图片，默认是 1000 张。

```
import os
import random

IMG_DIR = "/workspace/YOLOX/data/val2014"
max_image_count = 1000

save_list_name = os.path.join(IMG_DIR,"ImgList.txt")
image_list = os.listdir(IMG_DIR)
random.shuffle(image_list)
count_total = 0
str_write = ""
for idx, image_name in enumerate (image_list):
    if image_name[-3:] == "png"  or image_name[-3:] == "jpg" :
        count_total += 1
        line_str = image_name+" 0\n"
        str_write += line_str
    if count_total > max_image_count:
        break

with open(save_list_name,"w+") as fp:
    fp.write(str_write)
    fp.close()
```

② 、创建 lmdb 文件

使用脚本 create_lmdb.sh 创建 lmdb 文件，IMG_DIR 表示图片文件夹路径，执行完之后会在图片文件夹生成 img_lmdb 文件夹，即为 lmdb 文件。

```
#!/bin/bash

function convert_imageset_to_lmdb_demo()
{
```

```
  rm $IMG_DIR/img_lmdb -rif

  if [ ! -f $IMG_DIR/ImgList.txt ]; then echo "#ERROR: can not find
"$IMG_DIR"/ImgList.txt"; return; fi

  convert_imageset --shuffle --resize_height=640 --resize_width=640 \
    $IMG_DIR/  $IMG_DIR/ImgList.txt  $IMG_DIR/img_lmdb
  echo "#INFO: Convert Images to lmdb done"
  return $ret;
}

# IMG_DIR=./images
IMG_DIR=/workspace/YOLOX/data/val2014
convert_imageset_to_lmdb_demo
```

③ 、生成 fp32umodel

使用 yolox_to_fp32umodel.py 生成 fp32umodel，参数对应如下：

-m：输入 pytorch 路径

-s：网络的输入维度

-d：结果保存的文件夹

-D：lmdb 数据文件位置

```
import os
import ufw.tools as tools

pt_mobilenet = [
    '-m', '/workspace/YOLOX/models/yolox_s.pt',
    '-s', '(1,3,640,640)',
    '-d', '/workspace/YOLOX/models/yolox_s',
    '-D', '/workspace/YOLOX/data/val2017/img_lmdb',
    '--cmp'
]

if __name__ == '__main__':
    tools.pt_to_umodel(pt_mobilenet)
```

④ 、生成 int8umodel

在生成 fp32umodel 之后，会生成一个*bmnetp_test_fp32.prototxt，修改此文件，使网络最后的输出类型为 float，具体操作为在最后一层总添加“forward_with_float:true”。

```
layer {
  name: "15"
  type: "Transpose"
  bottom: "< 1 >137"
  top: "15"
  phase: TEST
  forward_with_float: true
  tag: "layer-1418@#aten::permute"
  transpose_param {
    order: 0
    order: 2
    order: 1
    order: 0
    order: 0
```

执行 convert_to_int8umodel.sh 生成 int8umodel，最终生成的 int8umodel 和 fp32umodel 在同一个文件夹下面。

```
calibration_use_pb \
    quantize \
    -model=/workspace/YOLOX/models/yolox_m/yolox_m_bmnetp_test_fp32.prototxt \
    -weights=/workspace/YOLOX/models/yolox_m/yolox_m_bmnetp.fp32umodel \
    -iterations=100 \
    -bitwidth=TO_INT8
```

⑤ 、生成 int8bmodel

执行 convert_to_int8bmodel.sh 生成 int8bmodel，其中 max_n 表述网络输入的最大 batch。

```
    bmnetu -model /workspace/YOLOX/models/yolox_m/yolox_m_bmnetp_deploy_int8_unique_top.prototxt \
    -weight /workspace/YOLOX/models/yolox_m/yolox_m_bmnetp.int8umodel \
    -max_n 4 \
    -prec=INT8 \
    -dyn=0 \
    -cmp=1 \
    -target=BM1684 \
    -outdir=/workspace/YOLOX/models/yolox_m/int8model
```

4、 推理代码

C++测试代码目录 examples/YOLOX/cpp_cv_bmcv_bmrt

5、 推理效率参考

1080p 解码+推理					
原始模型	硬件平台	运行模式	batch_size	推理效率	fps
yolox-s	SOC	fp32	4	126ms	31.7
		fp32	1	32ms	31.2
		int8	4	44.7ms	89.5
		int8	1	18.3ms	54.6
	PCIE	fp32	4	119ms	33.6

		fp32	1	30.9ms	32.4
		int8	4	38ms	105.2
		int8	1	17ms	58.8
yolox-m	SOC	fp32	4	254.5ms	15.7
		fp32	1	64.7ms	15.5
		int8	4	64.5ms	62
		int8	1	33ms	30.3
	PCIE	fp32	4	248ms	16.1
		fp32	1	63.5ms	15.7
		int8	4	58ms	68.9
		int8	1	32ms	31.2