CV180X CV181X Yolo 系列算法部署指南

Version: 1.0.1

Release date: 2023-07-25

©2022 北京晶视智能科技有限公司
本文件所含信息归北京晶视智能科技有限公司所有。
未经授权，严禁全部或部分复制或披露该等信息。

目录

修订记录

Revision	Date	Description
1.0.0	2023/7/25	初稿
1.0.1	2023/10/25	markdown 转 html

1 声明

法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。

本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

联系我们

地址

北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

电话

+86-10-57590723 +86-10-57590724

邮编

100094（北京）518100（深圳）

官方网站

https://www.sophgo.com/

技术论坛

https://developer.sophgo.com/forum/index.html

2 功能概述

2.1 目的

算能端侧提供的集成 yolo 系列算法 C++ 接口，用以缩短外部开发者定制化部署 yolo 系列模型所需的时间。

TDL_SDK 内部实现了 yolo 系列算法封装其前后处理和推理，提供统一且便捷的编程接口。

目前 TDL_SDK 包括但不限于 yolov5,yolov6,yolov7,yolov8,yolox,ppyoloe 等算法。

3 通用 yolov5 模型部署

3.1 引言

本文档介绍了如何将 yolov5 架构的模型部署在 cv181x 开发板的操作流程，主要的操作步骤包括：

- yolov5 模型 pytorch 版本转换为 onnx 模型
- onnx 模型转换为 cvimodel 格式
- 最后编写调用接口获取推理结果

3.2 pt 模型转换为 onnx

1. 首先可以下载 yolov5 官方仓库代码，地址如下：[ultralytics/yolov5: YOLOv5 🅵 in PyTorch > ONNX > CoreML > TFLite] (https://github.com/ultralytics/yolov5)

```
git clone https://github.com/ultralytics/yolov5.git
```

2. 获取 yolov5 的.pt 格式的模型，例如下载 yolov5s 模型的地址：[yolov5s](https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt)

3. 需要修改 yolov5/models/yolo.py 文件中 Detect 类中的 forward 函数，让 yolov5 的后处理由 RISC-V 来做，输出九个 branch，以后称这种为 TDL_SDK 导出方式

而原始输出为一个结果且后处理由模型，这种方式为官方导出结果。

- 原因是输出含有坐标范围比较大，容易量化失败或者效果很差。

将 yolo_export/yolov5_export.py 复制到 yolov5 仓库目录下

4. 然后使用以下命令导出 TDK_SDK 版本的 onnx 模型

```
#其中--weights表示权重文件的相对路径，--img-size表示输出尺寸为 640 x 640
python yolov5_export.py --weights ./weigthts/yolov5s.pt --img-size 640 640
#生成的onnx模型在当前目录
```

小技巧： 如果输入为 1080p 的视频流，建议将模型输入尺寸改为 384x640，可以减少冗余计算，提高推理速度，例如

```
python yolov5_export.py --weights ./weights/yolov5s.pt --img-size 384 640
```

3.3 准备转模型环境

onnx 转成 cvimodel 需要 TPU-MLIR 的发布包。TPU-MLIR 是算能 TDL 处理器的 TPU 编译器工程。

【TPU-MLIR 工具包下载】TPU-MLIR 代码路径 https://github.com/sophgo/tpu-mlir，感兴趣的可称为开源开发者共同维护开源社区。

而我们仅需要对应的工具包即可，下载地位为算能官网的 TPU-MLIR 论坛，后面简称为工具链工具包: (https://developer.sophgo.com/thread/473.html)

TPU-MLIR 工程提供了一套完整的工具链, 其可以将不同框架下预训练的神经网络, 转化为可以在算能 TPU 上高效运算的文件。

目前支持 onnx 和 Caffe 模型直接转换，其他框架的模型需要转换成 onnx 模型，再通过 TPU-MLIR 工具转换。

转换模型需要在指定的 docker 执行，主要的步骤可以分为两步：

- 第一步是通过 model_transform.py 将原始模型转换为 mlir 文件
- 第二步是通过 model_deploy.py 将 mlir 文件转换成 cvimodel

> 如果需要转换为 INT8 模型，还需要在第二步之前调用 run_calibration.py 生成校准表，然后传给 model_deploy.py

【Docker 配置】

TPU-MLIR 需要在 Docker 环境开发，可以直接下载 docker 镜像 (速度比较慢), 参考如下命令：

```
docker pull sophgo/tpuc_dev:latest
```

或者可以从【TPU 工具链工具包】中下载的 docker 镜像 (速度比较快)，然后进行加载 docker。

```
docker load -i  docker_tpuc_dev_v2.2.tar.gz
```

如果是首次使用 Docker，可以执行下述命令进行安装和配置（仅首次执行）：

```
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

【进入 docker 环境】确保安装包在当前目录，然后在当前目录创建容器如下：

```
docker run --privileged --name myname -v $PWD:/workspace -it sophgo/tpuc_dev:v2.2
```

后续的步骤假定用户当前处在 docker 里面的/workspace 目录

加载 tpu-mlir 工具包 & 准备工作目录

以下操作需要在 Docker 容器执行

【解压 tpu_mlir 工具包】以下文件夹创建主要是为了方便后续管理，也可按照自己喜欢的管理方式进行文件分类

新建一个文件夹 tpu_mlir，将新工具链解压到 tpu_mlir/目录下，并设置环境变量：

```
##其中tpu-mlir_xxx.tar.gz的xxx是版本号，根据对应的文件名而决定
mkdir tpu_mlir && cd tpu_mlir
cp tpu-mlir_xxx.tar.gz ./
tar zxf tpu-mlir_xxx.tar.gz
source tpu_mli_xxx/envsetup.sh
```

【拷贝 onnx 模型】创建一个文件夹，以 yolov5s 举例，创建一个文件夹 yolov5s，并将 onnx 模型放在 yolov5s/onnx/路径下

```
mkdir yolov5s && cd yolov5s
##上一节转出来的yolov5 onnx模型拷贝到yolov5s目录下
cp yolov5s.onnx ./
## 拷贝官网的dog.jpg过来做校验。
cp dog.jpg ./
```

上述准备工作完成之后，就可以开始转换模型

3.4 onnx 转 MLIR

如果模型是图片输入，在转模型之前我们需要了解模型的预处理。

如果模型用预处理后的 npz 文件做输入，则不需要考虑预处理。

本例子中 yolov5 的图片是 rgb,mean 和 scale 对应为：

- mean: 0.0, 0.0, 0.0
- scale: 0.0039216, 0.0039216, 0.0039216

模型转换的命令如下：

```
model_transform.py \
--model_name yolov5s \
--model_def yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--test_input ./dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s.mlir
```

其中 model_transform.py 参数详情，请参考【tpu_mlir_xxxxx/doc/TPU-MLIR 快速入门指南】

转换成 mlir 文件之后，会生成一个 yolov5s_in_f32.npz 文件，该文件是模型的输入文件

3.5 MLIR 转 INT8 模型

【生成校准表】

转 INT8 模型前需要跑 calibration，得到校准表；输入数据的数量根据情况准备 100~1000 张左右。

然后用校准表，生成 cvimodel. 生成校对表的图片尽可能和训练数据分布相似

```
## 这个数据集从COCO2017提取100来做校准，用其他图片也是可以的，这里不做强制要求。
run_calibration.py yolov5s.mlir \
--dataset COCO2017 \
--input_num 100 \
-o yolov5s_cali_table
```

运行完成之后会生成名为 yolov5_cali_table 的文件，该文件用于后续编译 cvimode 模型的输入文件

【生成 cvimodel】

然后生成 int8 对称量化 cvimodel 模型，执行如下命令：

其中–quant_output 参数表示将输出层也量化为 int8，不添加该参数则保留输出层为 float32。

从后续测试结果来说，将输出层量化为 int8，可以减少部分 ion，并提高推理速度，并且模型检测精度基本没有下降，推荐添加–quant_output 参数

```
model_deploy.py \
--mlir yolov5s.mlir \
--quant_input \
--quant_output \
--quantize INT8 \
--calibration_table yolov5s_cali_table \
--processor cv181x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--model yolov5_cv181x_int8_sym.cvimodel
```

其中 model_deploy.py 的主要参数参考, 请参考【tpu_mlir_xxxxx/doc/TPU-MLIR 快速入门指南】

编译完成后，会生成名为 yolov5_cv181x_int8_sym.cvimodel 的文件

在上述步骤运行成功之后，编译 cvimodel 的步骤就完成了，之后就可以使用 TDL_SDK 调用导出的 cvimodel 进行 yolov5 目标检测推理了。

> **小心：** 注意运行的对应平台要一一对应！

3.6 TDL_SDK 接口说明

tdlsdk 工具包需要联系算能获取。

集成的 yolov5 接口开放了预处理的设置，yolov5 模型算法的 anchor，conf 置信度以及 nms 置信度设置

预处理设置的结构体为 Yolov5PreParam

```
/** @struct YoloPreParam
 *  @ingroup core_cvitdlcore
 *  @brief Config the yolov5 detection preprocess.
 *  @var YoloPreParam::factor
 *  Preprocess factor, one dimension matrix, r g b channel
 *  @var YoloPreParam::mean
 *  Preprocess mean, one dimension matrix, r g b channel
 *  @var YoloPreParam::rescale_type
 *  Preprocess config, vpss rescale type config
 *  @var YoloPreParam::keep_aspect_ratio
 *  Preprocess config aspect scale
 *  @var YoloPreParam:: resize_method
 *  Preprocess resize method config
 *  @var YoloPreParam::format
 *  Preprocess pixcel format config
 */
typedef struct {
  float factor[3];
  float mean[3];
  meta_rescale_type_e rescale_type;
  bool keep_aspect_ratio;
  VPSS_SCALE_COEF_E resize_method;
  PIXEL_FORMAT_E format;
} YoloPreParam;
```

以下是一个简单的设置案例:

- 通过 CVI_TDL_Get_YOLO_Preparam 以及 CVI_TDL_Get_YOLO_Algparam 分别获取：初始化预处理设置 YoloPreParam 以及 yolov5 模型设置 YoloAlgParam
- 在设置了预处理参数和模型参数之后，再使用 CVI_TDL_Set_YOLO_Preparam 和 CVI_TDL_Set_YOLO_Algparam 传入设置的参数
  - yolov5 是 **anchor-based** 的检测算法，为了方便使用，开放了 anchor 自定义设置，在设置 YoloAlgParam 中，需要注意 anchors 和 strides 的顺序需要一一对应，否则会导致推理结果出现错误
  - 另外支持自定义分类数量修改，如果修改了模型的输出分类数量，需要设置 YolovAlgParam.cls 为修改后的分类数量
- 再打开模型 CVI_TDL_OpenModel
- 再打开模型之后可以设置对应的置信度和 nsm 阈值:
  - CVI_TDL_SetModelThreshold 设置置信度阈值，默认 0.5
  - CVI_TDL_SetModelNmsThreshold 设置 nsm 阈值，默认 0.5

```
// set preprocess and algorithm param for yolov5 detection
// if use official model, no need to change param
CVI_S32 init_param(const cvitdl_handle_t tdl_handle) {
  // setup preprocess
  YoloPreParam preprocess_cfg =
      CVI_TDL_Get_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5);

  for (int i = 0; i < 3; i++) {
    printf("asign val %d \n", i);
    preprocess_cfg.factor[i] = 0.003922;
    preprocess_cfg.mean[i] = 0.0;
  }
  preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("setup yolov5 param \n");
  CVI_S32 ret =
      CVI_TDL_Set_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5,F
↪preprocess_cfg);
  if (ret != CVI_SUCCESS) {
    printf("Can not set Yolov5 preprocess parameters %#x\n", ret);
    return ret;
  }

  // setup yolo algorithm preprocess
  YoloAlgParam yolov5_param = CVI_TDL_Get_YOLO_Algparam(tdl_handle, CVI_TDL_
↪SUPPORTED_MODEL_YOLOV5);
  uint32_t *anchors = new uint32_t[18];
  uint32_t p_anchors[18] = {10, 13, 16,  30,  33, 23,  30,  61,  62,
                            45, 59, 119, 116, 90, 156, 198, 373, 326};
  memcpy(anchors, p_anchors, sizeof(p_anchors));
  yolov5_param.anchors = anchors;
  yolov5_param.anchor_len = 18;

  uint32_t *strides = new uint32_t[3];
  uint32_t p_strides[3] = {8, 16, 32};
  memcpy(strides, p_strides, sizeof(p_strides));
  yolov5_param.strides = strides;
  yolov5_param.stride_len = 3;
  yolov5_param.cls = 80;

  printf("setup yolov5 algorithm param \n");
  ret = CVI_TDL_Set_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_
↪YOLOV5, yolov5_param);
  if (ret != CVI_SUCCESS) {
    printf("Can not set Yolov5 algorithm parameters %#x\n", ret);
    return ret;
  }

  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, 0.5);
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, 0.
↪5);

  printf("yolov5 algorithm parameters setup success!\n");
  return ret;
}
```

推理以及结果获取

通过本地或者流获取图片，并通过 CVI_TDL_ReadImage 函数读取图片，然后调用 Yolov5 推理接口 CVI_TDL_Yolov5。推理的结果存放在 obj_meta 结构体中，遍历获取边界框 bbox 的左上角以及右下角坐标点以及 object score(x1, y1, x2, y2, score)，另外还有分类 classes

```
ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, model_
↪path.c_str());
if (ret != CVI_SUCCESS) {
  printf("open model failed %#x!\n", ret);
  return ret;
}

// set thershold
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, 0.5);
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, 0.5);

std::cout << "model opened:" << model_path << std::endl;

VIDEO_FRAME_INFO_S fdFrame;
ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
std::cout << "CVI_TDL_ReadImage done!\n";

if (ret != CVI_SUCCESS) {
  std::cout << "Convert out video frame failed with :" << ret << ".file:" << str_src_dir
            << std::endl;
  // continue;
}

cvtdl_object_t obj_meta = {0};

CVI_TDL_Yolov5(tdl_handle, &fdFrame, &obj_meta);

for (uint32_t i = 0; i < obj_meta.size; i++) {
  printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_meta.info[i].bbox.y1,
         obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].bbox.score,
         obj_meta.info[i].classes);
}
```

3.7 编译说明

1. 下载 v4.x.x.x_source 压缩包，解压到当前文件夹 tar zxvf v4.x.x.x.x_source.tar.gz -C ./
2. 编译 v4.x ，注意设置为自己的开发版型号，开发版型号可以通过串口连接开发版，在开机的时候会有板子型号的相关信息输出

   使用以下命令编译:

   - 首先切换到解压后文件的目录
   - souce 环境变量
   - 定义开发版型号
   - 编译

```
cd v4.x.x.x_source/
source build/cvisetup.sh
defconfig CV1811h_wevb_0007a_spinor
build_all
```

3. 下载 TDL SDK 以及 TPU SDK，将其解压到 v4.x.x.x_source/install/soc_cv1811h_wevb_0007a_spinor/tpu_musl_riscv64/目录下

```
tar ai_sdk.tar.gz -C /v4.x.x.x_source/install/soc_cv1811h_wevb_0007a_spinor/tpu_musl_
↪riscv64/
tar tpu_sdk.tar.gz -C /v4.x.x.x_source/install/soc_cv1811h_wevb_0007a_spinor/tpu_musl_
↪riscv64/
```

4. 切换到 ai sdk/sample/cvi_tdl/目录下，然后输入以下命令编译可执行的 sample 程序

```
make MW_PATH=../../../../../../middleware/v2/ \
    TPU_PATH=../cvitek_tpu_sdk/ \
    USE_TPU_IVE=OFF \
    CHIP=CV181X \
    SDK_VER=musl_riscv64

make install
```

若想清理生成的文件，可以执行

```
make clean
```

小心： 注意依赖路径是否正确！CHIP 要设置为自己使用的芯片型号，芯片参数全部为大写！

5. 编译完成之后，可以连接开发板并执行程序：
   - 开发板连接网线，确保开发板和电脑在同一个网关
   - 电脑通过串口连接开发板，波特率设置为 115200，电脑端在串口输入 ifconfig 获取开发板的 ip 地址
   - 电脑通过 ssh 远程工具连接对应 ip 地址的开发板，用户名默认为：root，密码默认为：cvitek_tpu_sdk
   - **连接开发板之后，可以通过 mount 挂在 sd 卡或者电脑的文件夹：**
     - 改载 sd 卡的命令是

```
mount /dev/mmcblk0 /mnt/sd
# or
mount /dev/mmcblk0p1 /mnt/sd
```

     - 挂载电脑的命令是：

```
mount -t nfs 10.80.39.3:/sophgo/nfsuser ./admin1_data -o nolock
```

     主要修改 ip 地址为自己电脑的 ip，路径同样修改为自己的路径

6. export 动态依赖库

   主要需要的动态依赖库为：

   - ai_sdk 目录下的 lib
   - tpu_sdk 目录下的 lib
   - middlewave/v2/lib
   - middleware/v2/3rd
   - ai_sdk 目录下的 sample/3rd/lib

示例如下：

```
export LD_LIBRARY_PATH=/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_
↪0007a_spinor/tpu_musl_riscv64/cvitek_ive_sdk/lib:\
/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_0007a_spinor/rootfs/mnt/
↪system/usr/lib:\
/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_0007a_spinor/tpu_musl_
↪riscv64/cvitek_ai_sdk/lib:\
/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_0007a_spinor/tpu_musl_
↪riscv64/cvitek_tpu_sdk/lib:\
/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_0007a_spinor/rootfs/mnt/
↪system/usr/lib/3rd:\
/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_0007a_spinor/rootfs/mnt/
↪system/usr/lib:\
/path/to/v4.x.x.x/on/board/middleware/v2/lib:\
/path/to/v4.x.x.x/on/board/middleware/v2/lib/3rd:\
/path/to/v4.x.x.x/on/board/install/soc_cv1811h_wevb_0007a_spinor/tpu_musl_
↪riscv64/cvitek_ai_sdk/sample/3rd
```

小心： 注意将/path/to/v4.x.x.x/on/board/修改为开发版可以访问的路径，如果是用 sd 卡挂载，可以提前将所有需要的 lib 目录下的文件拷贝在同一个文件夹，然后 export 对应在 sd 卡的路径即可

7. 运行 sample 程序

- 切换到挂载的 ai_sdk/sample/tmp_install/目录下
- 然后运行以下测试案例

```
./sample_yolov5 /path/to/yolov5s.cvimodel /path/to/test.jpg
```

上述运行命令注意选择自己的 cvimodel 以及测试图片的挂载路径

3.8 测试结果

以下是官方 yolov5 模型转换后在 coco2017 数据集测试的结果，测试平台为 **CV1811h_wevb_0007a_spinor**

以下测试使用阈值为：

- conf_thresh: 0.001
- nms_thresh: 0.65

输入分辨率均为 640 x 640

yolov5s 模型的官方导出导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MB	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	56.8	37.4
cv181x	92.8	100.42	16.01	量化失败	量化失败
cv182x	69.89	102.74	16	量化失败	量化失败
cv183x	25.66	73.4	N/A	量化失败	量化失败

yolov5s 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MB	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	55.4241	36.6361
cv181x	87.76	85.74	15.8	54.204	34.3985
cv182x	65.33	87.99	15.77	54.204	34.3985
cv183x	22.86	58.38	14.22	54.204	34.3985

yolov5m 模型的官方导出导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MB	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	64.1	45.4
cv181x	ion 分配失败	ion 分配失败	35.96	量化失败	量化失败
cv182x	180.85	258.41	35.97	量化失败	量化失败
cv183x	59.36	137.86	30.49	量化失败	量化失败

yolov5m 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MB	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	62.770	44.4973
cv181x	N/A	N/A	35.73	ion 分配失败	ion 分配失败
cv182x	176.04	243.62	35.74	61.5907	42.0852
cv183x	56.53	122.9	30.27	61.5907	42.0852

4 通用 yolov6 模型部署

4.1 引言

本文档介绍了如何将 yolov6 架构的模型部署在 cv181x 开发板的操作流程，主要的操作步骤包括：

- yolov6 模型 pytorch 版本转换为 onnx 模型
- onnx 模型转换为 cvimodel 格式
- 最后编写调用接口获取推理结果

4.2 pt 模型转换为 onnx

下载 yolov6 官方仓库 [meituan/YOLOv6](https://github.com/meituan/YOLOv6)，下载 yolov6 权重文件，在 yolov6 文件夹下新建一个目录 weights，并将下载的权重文件放在目录 yolov6-main/weights/下

然后将 yolo_export/yolov6_eport.py 复制到 yolov6-main/deploy/onnx 目录下

通过以下命令导出 onnx 模型

```
python ./deploy/ONNX/yolov6_export.py \
  --weights ./weights/yolov6n.pt \
  --img-size 640 640
```

- weights 为 pytorch 模型文件的路径
- img-size 为模型输入尺寸

然后得到 onnx 模型

小技巧: 如果输入为 1080p 的视频流，建议将模型输入尺寸改为 384x640，可以减少冗余计算，提高推理速度，如下所示

```
python yolov6_export.py --weights path/to/pt/weights --img-size 384 640
```

4.3 onnx 模型转换 cvimodel

cvimodel 转换操作可以参考 yolo-v5 移植章节的 onnx 模型转换 cvimodel 部分。

4.4 yolov6 接口说明

提供预处理参数以及算法参数设置，其中参数设置：

- YoloPreParam 输入预处理设置

  $y=(x-mean)times factor$

  - factor 预处理方差的倒数
  - mean 预处理均值
  - use_quantize_scale 预处理图片尺寸
  - format 图片格式
- YoloAlgParam
  - cls 设置 yolov6 模型的分类

> yolov6 是 anchor-free 的目标检测网络，不需要传入 anchor

另外是 yolov6 的两个参数设置：

- CVI_TDL_SetModelThreshold 设置置信度阈值，默认为 0.5
- CVI_TDL_SetModelNmsThreshold 设置 nms 阈值，默认为 0.5

```
// set preprocess and algorithm param for yolov6 detection
// if use official model, no need to change param
CVI_S32 init_param(const cvitdl_handle_t tdl_handle) {
  // setup preprocess
  YoloPreParam preprocess_cfg =
      CVI_TDL_Get_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6);

  for (int i = 0; i < 3; i++) {
    printf("asign val %d \n", i);
    preprocess_cfg.factor[i] = 0.003922;
    preprocess_cfg.mean[i] = 0.0;
  }
  preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("setup yolov6 param \n");
  CVI_S32 ret =
      CVI_TDL_Set_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6,F
↪preprocess_cfg);
  if (ret != CVI_SUCCESS) {
    printf("Can not set yolov6 preprocess parameters %#x\n", ret);
    return ret;
  }
```

(续下页)

(接上页)

```
  // setup yolo algorithm preprocess
  YoloAlgParam yolov6_param = CVI_TDL_Get_YOLO_Algparam(tdl_handle, CVI_TDL_
↪SUPPORTED_MODEL_YOLOV6);
  yolov6_param.cls = 80;

  printf("setup yolov6 algorithm param \n");
  ret = CVI_TDL_Set_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_
↪YOLOV6, yolov6_param);
  if (ret != CVI_SUCCESS) {
    printf("Can not set yolov6 algorithm parameters %#x\n", ret);
    return ret;
  }

  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6, 0.5);
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6, 0.
↪5);

  printf("yolov6 algorithm parameters setup success!\n");
  return ret;
}
```

推理代码如下：

推理代码如下：

通过本地或者流获取图片，并通过 CVI_TDL_ReadImage 函数读取图片，然后调用 Yolov6 推理接口 CVI_TDL_Yolov6。推理的结果存放在 obj_meta 结构体中，遍历获取边界框 bbox 的左上角以及右下角坐标点以及 object score(x1, y1, x2, y2, score)，另外还有分类 classes

```
ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6, model_
↪path.c_str());
if (ret != CVI_SUCCESS) {
  printf("open model failed %#x!\n", ret);
  return ret;
}
printf("cvimodel open success!\n");

std::cout << "model opened:" << model_path << std::endl;

VIDEO_FRAME_INFO_S fdFrame;
ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
std::cout << "CVI_TDL_ReadImage done!\n";

if (ret != CVI_SUCCESS) {
  std::cout << "Convert out video frame failed with :" << ret << ".file:" << str_src_dir
            << std::endl;
}

cvtdl_object_t obj_meta = {0};

CVI_TDL_Yolov6(tdl_handle, &fdFrame, &obj_meta);

printf("detect number: %d\n", obj_meta.size);
```

(续下页)

(接上页)

```
for (uint32_t i = 0; i < obj_meta.size; i++) {
  printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_meta.info[i].bbox.y1,
         obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].bbox.score,
         obj_meta.info[i].classes);
}
```

4.5 测试结果

转换了 yolov6 官方仓库给出的 yolov6n 以及 yolov6s，测试数据集为 COCO2017

其中阈值参数设置为：

- conf_threshold: 0.03
- nms_threshold: 0.65

分辨率均为 640x640

yolov6n 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	53.1	37.5
cv181x	ion 分配失败	ion 分配失败	11.58	量化失败	量化失败
cv182x	39.17	47.08	11.56	量化失败	量化失败
cv183x	量化失败	量化失败	量 化失败	量化失败	量化失败

yolov6n 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	51.6373	36.4384
cv181x	49.11	31.35	8.46	49.8226	34.284
cv182x	34.14	30.53	8.45	49.8226	34.284
cv183x	10.89	21.22	8.49	49.8226	34.284

yolov6s 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	61.8	45
cv181x	ion 分配失败	ion 分配失败	27.56	量化失败	量化失败
cv182x	131.1	115.81	27.56	量化失败	量化失败
cv183x	量化失败	量化失败	量 化失败	量化失败	量化失败

yolov6s 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MI	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	60.1657	43.5878
cv181x	ion 分配失败	ion 分配失败	25.33	ion 分配失败	ion 分配失败
cv182x	126.04	99.16	25.32	56.2774	40.0781
cv183x	38.55	57.26	23.59	56.2774	40.0781

5 通用 yolov7 模型部署

5.1 引言

本文档介绍了如何将 yolov7 架构的模型部署在 cv181x 开发板的操作流程，主要的操作步骤包括：

- yolov7 模型 pytorch 版本转换为 onnx 模型
- onnx 模型转换为 cvimodel 格式
- 最后编写调用接口获取推理结果

5.2 pt 模型转换为 onnx

下载官方 [yolov7](https://github.com/WongKinYiu/yolov7) 仓库代码

```
git clone https://github.com/WongKinYiu/yolov7.git
```

在上述下载代码的目录中新建一个文件夹 weights，然后将需要导出 onnx 的模型移动到 yolov7/weights

```
cd yolov7 & mkdir weights
cp path/to/onnx ./weights/
```

然后将 yolo_export/yolov7_export.py 复制到 yolov7 目录下

然后使用以下命令导出 TDL_SDK 形式的 yolov7 模型

```
python yolov7_export.py --weights ./weights/yolov7-tiny.pt
```

小技巧： 如果输入为 1080p 的视频流，建议将模型输入尺寸改为 384x640，可以减少冗余计算，提高推理速度，如下命令所示：

```
python yolov7_export.py --weights ./weights/yolov7-tiny.pt --img-size 384 640
```

5.3 onnx 模型转换 cvimodel

cvimodel 转换操作可以参考 yolo-v5 移植章节的 onnx 模型转换 cvimodel 部分。

小心： yolov7 官方版本的模型预处理参数，即 mean 以及 scale 与 yolov5 相同，可以复用 yolov5 转换 cvimodel 的命令

5.4 TDL_SDK 接口说明

yolov7 模型与 yolov5 模型检测与解码过程基本类似，主要不同是 anchor 的不同

小心： **注意修改 anchors 为 yolov7 的 anchors!!!**

anchors:

- [12,16, 19,36, 40,28] # P3/8
- [36,75, 76,55, 72,146] # P4/16
- [142,110, 192,243, 459,401] # P5/32

预处理接口设置如下代码所示

```
// set preprocess and algorithm param for yolov7 detection
// if use official model, no need to change param
CVI_S32 init_param(const cvitdl_handle_t tdl_handle) {
  // setup preprocess
  YoloPreParam preprocess_cfg =
      CVI_TDL_Get_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV7);

  for (int i = 0; i < 3; i++) {
    printf("asign val %d \n", i);
    preprocess_cfg.factor[i] = 0.003922;
    preprocess_cfg.mean[i] = 0.0;
  }
  preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("setup yolov7 param \n");
  CVI_S32 ret =
      CVI_TDL_Set_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV7,F
↪preprocess_cfg);
  if (ret != CVI_SUCCESS) {
    printf("Can not set Yolov5 preprocess parameters %#x\n", ret);
    return ret;
  }

  // setup yolo algorithm preprocess
  YoloAlgParam yolov7_param = CVI_TDL_Get_YOLO_Algparam(tdl_handle, CVI_TDL_
↪SUPPORTED_MODEL_YOLOV7);
```

(续下页)

(接上页)

```
  uint32_t *anchors = new uint32_t[18];
  uint32_t p_anchors[18] = {12, 16, 19, 36, 40, 28, 36, 75, 76,
                            55, 72, 146, 142, 110, 192, 243, 459, 401};
  memcpy(anchors, p_anchors, sizeof(p_anchors));
  yolov7_param.anchors = anchors;

  uint32_t *strides = new uint32_t[3];
  uint32_t p_strides[3] = {8, 16, 32};
  memcpy(strides, p_strides, sizeof(p_strides));
  yolov7_param.strides = strides;
  yolov7_param.cls = 80;

  printf("setup yolov7 algorithm param \n");
  ret = CVI_TDL_Set_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_
↪YOLOV7, yolov7_param);
  if (ret != CVI_SUCCESS) {
    printf("Can not set Yolov5 algorithm parameters %#x\n", ret);
    return ret;
  }

  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV7, 0.5);
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV7, 0.
↪5);

  printf("yolov7 algorithm parameters setup success!\n");
  return ret;
}
```

推理接口如下所示：

```
ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV7, model_
↪path.c_str());
if (ret != CVI_SUCCESS) {
  printf("open model failed %#x!\n", ret);
  return ret;
}

std::cout << "model opened:" << model_path << std::endl;

VIDEO_FRAME_INFO_S fdFrame;
ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
std::cout << "CVI_TDL_ReadImage done!\n";

if (ret != CVI_SUCCESS) {
  std::cout << "Convert out video frame failed with :" << ret << ".file:" << str_src_dir
            << std::endl;
  // continue;
}

cvtdl_object_t obj_meta = {0};

CVI_TDL_Yolov7(tdl_handle, &fdFrame, &obj_meta);
```

(续下页)

(接上页)

```
for (uint32_t i = 0; i < obj_meta.size; i++) {
  printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_meta.info[i].bbox.y1,
         obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].bbox.score,
         obj_meta.info[i].classes);
}
```

5.5 测试结果

测试了 yolov7-tiny 模型各个版本的指标，测试数据为 COCO2017，其中阈值设置为：

- conf_threshold: 0.001
- nms_threshold: 0.65

分辨率均为 640 x 640

yolov7-tiny 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	56.7	38.7
cv181x	75.4	85.31	17.54	量化失败	量化失败
cv182x	56.6	85.31	17.54	量化失败	量化失败
cv183x	21.85	71.46	16.15	量化失败	量化失败

yolov7-tiny 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	53.7094	36.438
cv181x	70.41	70.66	15.43	53.3681	32.6277
cv182x	52.01	70.66	15.43	53.3681	32.6277
cv183x	18.95	55.86	14.05	53.3681	32.6277

6 通用 yolov8 模型部署

6.1 引言

本文档介绍了如何将 yolov8 架构的模型部署在 cv181x 开发板的操作流程，主要的操作步骤包括：

- yolov8 模型 pytorch 版本转换为 onnx 模型
- onnx 模型转换为 cvimodel 格式
- 最后编写调用接口获取推理结果

6.2 pt 模型转换为 onnx

首先获取 yolov8 官方仓库代码 [ultralytics/ultralytics: NEW - YOLOv8 🅵 in PyTorch > ONNX > OpenVINO > CoreML > TFLite (github.com)](https://github.com/ultralytics/ultralytics)

```
git clone https://github.com/ultralytics/ultralytics.git
```

再下载对应的 yolov8 模型文件，以 [yolov8n](https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt) 为例，然后将下载的 yolov8n.pt 放在 ultralytics/weights/目录下，如下命令行所示

```
cd ultralytics & mkdir weights
cd weights
wget https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt
```

调整 yolov8 输出分支，去掉 forward 函数的解码部分，并将三个不同的 feature map 的 box 以及 cls 分开，得到 6 个分支

将 yolo_export/yolov8_export.py 代码复制到 yolov8 仓库下，然后使用以下命令导出分支版本的 onnx 模型：

```
python yolov8_export.py --weights ./weights/yolov8.pt
```

运行上述代码之后，可以在./weights/目录下得到 yolov8n.onnx 文件，之后就是将 onnx 模型转换为 cvimodel 模型

小技巧: 如果输入为 1080p 的视频流，建议将模型输入尺寸改为 384x640，可以减少冗余计算，提高推理速度，如下：

```
python yolov8_export.py --weights ./weights/yolov8.pt --img-size 384 640
```

6.3 onnx 模型转换 cvimodel

cvimodel 转换操作可以参考 cvimodel 转换操作可以参考 yolo-v5 移植章节的 onnx 模型转换 cvimodel 部分。

6.4 TDL_SDK 接口说明

yolov8 的预处理设置参考如下：

```
// set preprocess and algorithm param for yolov8 detection
// if use official model, no need to change param
CVI_S32 init_param(const cvitdl_handle_t tdl_handle) {
  // setup preprocess
  YoloPreParam preprocess_cfg =
      CVI_TDL_Get_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
↪DETECTION);

  for (int i = 0; i < 3; i++) {
    printf("asign val %d \n", i);
    preprocess_cfg.factor[i] = 0.003922;
    preprocess_cfg.mean[i] = 0.0;
  }
  preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("setup yolov8 param \n");
  CVI_S32 ret = CVI_TDL_Set_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_
↪MODEL_YOLOV8_DETECTION,
                                          preprocess_cfg);
  if (ret != CVI_SUCCESS) {
    printf("Can not set yolov8 preprocess parameters %#x\n", ret);
    return ret;
  }

  // setup yolo algorithm preprocess
  YoloAlgParam yolov8_param =
      CVI_TDL_Get_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
↪DETECTION);
  yolov8_param.cls = 80;

  printf("setup yolov8 algorithm param \n");
  ret =
      CVI_TDL_Set_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
```

(续下页)

(接上页)

```
↪DETECTION, yolov8_param);
 if (ret != CVI_SUCCESS) {
   printf("Can not set yolov8 algorithm parameters %#x\n", ret);
   return ret;
 }

 // set theshold
 CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
↪DETECTION, 0.5);
 CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
↪DETECTION, 0.5);

 printf("yolov8 algorithm parameters setup success!\n");
 return ret;
}
```

推理测试代码：

```
ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
↪DETECTION, argv[1]);

if (ret != CVI_SUCCESS) {
  printf("open model failed with %#x!\n", ret);
  return ret;
}
printf("--------------------to do detection----------------------\n");

VIDEO_FRAME_INFO_S bg;
ret = CVI_TDL_ReadImage(strf1.c_str(), &bg, PIXEL_FORMAT_RGB_888_PLANAR);
if (ret != CVI_SUCCESS) {
  printf("open img failed with %#x!\n", ret);
  return ret;
} else {
  printf("image read,width:%d\n", bg.stVFrame.u32Width);
  printf("image read,hidth:%d\n", bg.stVFrame.u32Height);
}
std::string str_res;
cvtdl_object_t obj_meta = {0};
CVI_TDL_YOLOV8_Detection(tdl_handle, &bg, &obj_meta);

std::cout << "objnum:" << obj_meta.size << std::endl;
std::stringstream ss;
ss << "boxes=[";
for (uint32_t i = 0; i < obj_meta.size; i++) {
  ss << "[" << obj_meta.info[i].bbox.x1 << "," << obj_meta.info[i].bbox.y1 << ","
    << obj_meta.info[i].bbox.x2 << "," << obj_meta.info[i].bbox.y2 << ","
    << obj_meta.info[i].classes << "," << obj_meta.info[i].bbox.score << "],";
}
ss << "]\n";
std::cout << ss.str();
```

6.5 测试结果

转换测试了官网的 yolov8n 以及 yolov8s 模型，在 COCO2017 数据集上进行了测试，其中阈值设置为：

- conf: 0.001
- nms_thresh: 0.6

所有分辨率均为 640 x 640

yolov8n 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	53	37.3
cv181x	54.91	44.16	8.64	量化失败	量化失败
cv182x	40.21	44.32	8.62	量化失败	量化失败
cv183x	17.81	40.46	8.3	量化失败	量化失败

yolov8n 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	51.32	36.4577
cv181x	45.62	31.56	7.54	51.2207	35.8048
cv182x	32.8	32.8	7.72	51.2207	35.8048
cv183x	12.61	28.64	7.53	51.2207	35.8048

yolov8s 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	61.8	44.9
cv181x	144.72	101.75	17.99	量化失败	量化失败
cv182x	103	101.75	17.99	量化失败	量化失败
cv183x	38.04	38.04	16.99	量化失败	量化失败

yolov8s 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	60.1534	44.034
cv181x	135.55	89.53	18.26	60.2784	43.4908
cv182x	95.95	89.53	18.26	60.2784	43.4908
cv183x	32.88	58.44	16.9	60.2784	43.4908

7 通用 yolox 模型部署

7.1 引言

本文档介绍了如何将 yolox 架构的模型部署在 cv181x 开发板的操作流程，主要的操作步骤包括：

- yolox 模型 pytorch 版本转换为 onnx 模型
- onnx 模型转换为 cvimodel 格式
- 最后编写调用接口获取推理结果

7.2 pt 模型转换为 onnx

首先可以在 github 下载 yolox 的官方代码：[Megvii-BaseDetection/YOLOX: YOLOX is a high-performance anchor-free YOLO, exceeding yolov3~v5 with MegEngine, ONNX, TensorRT, ncnn, and OpenVINO supported. Documentation: https://yolox.readthedocs.io/ (github.com)](https://github.com/Megvii-BaseDetection/YOLOX/tree/main)

使用以下命令从源代码安装 YOLOX

```
git clone git@github.com:Megvii-BaseDetection/YOLOX.git
cd YOLOX
pip3 install -v -e .  # or  python3 setup.py develop
```

需要切换到刚刚下载的 YOLOX 仓库路径，然后创建一个 weights 目录，将预训练好的.pth 文件移动至此

```
cd YOLOX & mkdir weights
cp path/to/pth ./weigths/
```

【官方导出 onnx】

切换到 tools 路径

```
cd tools
```

在 onnx 中解码的导出方式

```
python \
export_onnx.py \
--output-name ../weights/yolox_m_official.onnx \
-n yolox-m \
--no-onnxsim \
-c ../weights/yolox_m.pth \
--decode_in_inference
```

相关参数含义如下：

- –output-name 表示导出 onnx 模型的路径和名称
- -n 表示模型名，可以选择 * yolox-s, m, l, x * yolo-nano * yolox-tiny * yolov3
- -c 表示预训练的.pth 模型文件路径
- –decode_in_inference 表示是否在 onnx 中解码

【TDL_SDK 版本导出 onnx】

为了保证量化的精度，需要将 YOLOX 解码的 head 分为三个不同的 branch 输出，而不是官方版本的合并输出

通过以下的脚本和命令导出三个不同 branch 的 head：

将 yolo_export/yolox_export.py 复制到 YOLOX/tools 目录下，然后使用以下命令导出分支输出的 onnx 模型：

```
python \
yolox_export.py \
--output-name ../weights/yolox_s_9_branch_384_640.onnx \
-n yolox-s \
-c ../weights/yolox_s.pth
```

小技巧： 如果输入为 1080p 的视频流，建议将模型输入尺寸改为 384x640，可以减少冗余计算，提高推理速度，如下：

```
python \
yolox_export.py \
--output-name ../weights/yolox_s_9_branch_384_640.onnx \
-n yolox-s \
-c ../weights/yolox_s.pth \
--img-size 384 640
```

7.3 onnx 模型转换 cvimodel

cvimodel 转换操作可以参考 yolo-v5 移植章节的 onnx 模型转换 cvimodel 部分。

7.4 TDL_SDK 接口说明

预处理参数设置

预处理参数设置通过一个结构体传入设置参数

```
typedef struct {
  float factor[3];
  float mean[3];
  meta_rescale_type_e rescale_type;

  bool use_quantize_scale;
  PIXEL_FORMAT_E format;
} YoloPreParam;
```

而对于 YOLOX，需要传入以下四个参数：

- factor 预处理 scale 参数
- mean 预处理均值参数
- use_quantize_scale 是否使用模型的尺寸，默认为 true
- format 图片格式，PIXEL_FORMAT_RGB_888_PLANAR

其中预处理 factor 以及 mean 的公式为 $$ y=(x-mean)times scale $$

算法参数设置

```
typedef struct {
  uint32_t cls;
} YoloAlgParam;
```

需要传入分类的数量，例如

```
YoloAlgParam p_yolo_param;
p_yolo_param.cls = 80;
```

另外的模型置信度参数设置以及 NMS 阈值设置如下所示：

```
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, conf_
↪threshold);
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, nms_
↪threshold);
```

其中 conf_threshold 为置信度阈值；nms_threshold 为 nms 阈值

测试代码

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <chrono>
#include <fstream>
#include <functional>
```

(续下页)

(接上页)

```
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>
#include "core.hpp"
#include "core/cvi_tdl_types_mem_internal.h"
#include "core/utils/vpss_helper.h"
#include "cvi_tdl.h"
#include "cvi_tdl_media.h"
#include "sys_utils.hpp"

int main(int argc, char* argv[]) {
  int vpssgrp_width = 1920;
  int vpssgrp_height = 1080;
  CVI_S32 ret = MMF_INIT_HELPER2(vpssgrp_width, vpssgrp_height, PIXEL_FORMAT_RGB_
↪888, 1,
                                 vpssgrp_width, vpssgrp_height, PIXEL_FORMAT_RGB_888, 1);
  if (ret != CVI_TDL_SUCCESS) {
    printf("Init sys failed with %#x!\n", ret);
    return ret;
  }

  cvitdl_handle_t tdl_handle = NULL;
  ret = CVI_TDL_CreateHandle(&tdl_handle);
  if (ret != CVI_SUCCESS) {
    printf("Create tdl handle failed with %#x!\n", ret);
    return ret;
  }
  printf("start yolox preprocess config \n");
  // // setup preprocess
  YoloPreParam p_preprocess_cfg;

  for (int i = 0; i < 3; i++) {
    p_preprocess_cfg.factor[i] = 1.0;
    p_preprocess_cfg.mean[i] = 0.0;
  }
  p_preprocess_cfg.use_quantize_scale = true;
  p_preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("start yolo algorithm config \n");
  // setup yolo param
  YoloAlgParam p_yolo_param;
  p_yolo_param.cls = 80;

  printf("setup yolox param \n");
  ret = CVI_TDL_Set_YOLOX_Param(tdl_handle, &p_preprocess_cfg, &p_yolo_param);
  printf("yolox set param success!\n");
  if (ret != CVI_SUCCESS) {
    printf("Can not set YoloX parameters %#x\n", ret);
    return ret;
  }

  std::string model_path = argv[1];
```

(续下页)

(接上页)

```
  std::string str_src_dir = argv[2];

  float conf_threshold = 0.5;
  float nms_threshold = 0.5;
  if (argc > 3) {
    conf_threshold = std::stof(argv[3]);
  }

  if (argc > 4) {
    nms_threshold = std::stof(argv[4]);
  }

  printf("start open cvimodel...\n");
  ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, model_
↪path.c_str());
  if (ret != CVI_SUCCESS) {
    printf("open model failed %#x!\n", ret);
    return ret;
  }
  printf("cvimodel open success!\n");
  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, conf_
↪threshold);
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX,F
↪nms_threshold);

  std::cout << "model opened:" << model_path << std::endl;

  VIDEO_FRAME_INFO_S fdFrame;
  ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
  std::cout << "CVI_TDL_ReadImage done!\n";

  if (ret != CVI_SUCCESS) {
    std::cout << "Convert out video frame failed with :" << ret << ".file:" << str_src_dir
              << std::endl;
  }

  cvtdl_object_t obj_meta = {0};

  CVI_TDL_YoloX(tdl_handle, &fdFrame, &obj_meta);

  printf("detect number: %d\n", obj_meta.size);
  for (uint32_t i = 0; i < obj_meta.size; i++) {
    printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_meta.info[i].bbox.y1,
           obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].bbox.score,
           obj_meta.info[i].classes);
  }

  CVI_VPSS_ReleaseChnFrame(0, 0, &fdFrame);
  CVI_TDL_Free(&obj_meta);
  CVI_TDL_DestroyHandle(tdl_handle);

  return ret;
}
```

7.5 测试结果

测试了 yolox 模型 onnx 以及在 cv181x/2x/3x 各个平台的性能指标，其中参数设置：

- conf: 0.001
- nms: 0.65
- 分辨率：640 x 640

yolox-s 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	59.3	40.5
cv181x	131.95	104.46	16.43	量化失败	量化失败
cv182x	95.75	104.85	16.41	量化失败	量化失败
cv183x	量化失败	量化失败	量 化失败	量化失败	量化失败

yolox-s 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	53.1767	36.4747
cv181x	127.91	95.44	16.24	52.4016	35.4241
cv182x	91.67	95.83	16.22	52.4016	35.4241
cv183x	30.6	65.25	14.93	52.4016	35.4241

yolox-m 模型的官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(ME	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	65.6	46.9
cv181x	ion 分配失败	ion 分配失败	39.18	量化失败	量化失败
cv182x	246.1	306.31	39.16	量化失败	量化失败
cv183x	量化失败	量化失败	量 化失败	量化失败	量化失败

yolox-m 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MB	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	59.9411	43.0057
cv181x	ion 分配失败	ion 分配失败	38.95	59.3559	42.1688
cv182x	297.5	242.65	38.93	59.3559	42.1688
cv183x	75.8	144.97	33.5	59.3559	42.1688

8 通用 pp-yoloe 模型部署

8.1 引言

本文档介绍了如何将 ppyoloe 架构的模型部署在 cv181x 开发板的操作流程，主要的操作步骤包括：

- ppyoloe 模型 pytorch 版本转换为 onnx 模型
- onnx 模型转换为 cvimodel 格式
- 最后编写调用接口获取推理结果

8.2 pt 模型转换为 onnx

PP-YOLOE 是基于 PP-Yolov2 的 Anchor-free 模型，官方仓库在 [PaddleDetection](https://github.com/PaddlePaddle/PaddleDetection)

获取官方仓库代码并安装：

```
git clone https://github.com/PaddlePaddle/PaddleDetection.git

# CUDA10.2
python -m pip install paddlepaddle-gpu==2.3.2 -i https://mirror.baidu.com/pypi/simple
```

其他版本参照官方安装文档 [开始使用 _ 飞桨-源于产业实践的开源深度学习平台 (paddlepaddle.org.cn)](https://www.paddlepaddle.org.cn/install/quick?docurl=/documentation/docs/zh/install/pip/linux-pip.html)

onnx 导出可以参考官方文档 [PaddleDetection/deploy/EXPORT_ONNX_MODEL.md at release/2.4 · PaddlePaddle/PaddleDetection (github.com)](https://github.com/PaddlePaddle/PaddleDetection/blob/release/2.4/deploy/EXPORT_ONNX_MODEL.md)

本文档提供官方版本直接导出方式以及算能版本导出 onnx，算能版本导出的方式需要去掉检测头的解码部分，方便后续量化，解码部分交给 TDL_SDK 实现

【官方版本导出】

可以使用 PaddleDetection/tools/export_model.py 导出官方版本的 onnx 模型

使用以下命令可以实现自动导出 onnx 模型，导出的 onnx 模型路径在 output_inference_official/ppyoloe_crn_s_300e_coco/ppyoloe_crn_s_300e_coco_official.onnx

```
cd PaddleDetection
python \
tools/export_model_official.py \
-c configs/ppyoloe/ppyoloe_crn_s_300e_coco.yml \
-o weights=https://paddledet.bj.bcebos.com/models/ppyoloe_crn_s_300e_coco.pdparams

paddle2onnx \
--model_dir \
output_inference/ppyoloe_crn_s_300e_coco \
--model_filename model.pdmodel \
--params_filename model.pdiparams \
--opset_version 11 \
--save_file output_inference_official/ppyoloe_crn_s_300e_coco/ppyoloe_crn_s_300e_coco_official.
↪onnx
```

参数说明：

- -c 模型配置文件
- -o paddle 模型权重
- –model_dir 模型导出目录
- –model_filename paddle 模型的名称
- –params_filename paddle 模型配置
- –opset_version opset 版本配置
- –save_file 导出 onnx 模型的相对路径

【算能版本导出】

为了更好地进行模型量化，需要将检测头解码的部分去掉，再导出 onnx 模型，使用以下方式导出不解码的 onnx 模型

将 yolo_export/pp_yolo_export.py 复制到 tools/目录下，然后使用如下命令导出不解码的 pp-yoloe 的 onnx 模型

```
python \
tools/export_model_no_decode.py \
-c configs/ppyoloe/ppyoloe_crn_s_300e_coco.yml \
-o weights=https://paddledet.bj.bcebos.com/models/ppyoloe_crn_s_300e_coco.pdparams

paddle2onnx \
--model_dir \
output_inference/ppyoloe_crn_s_300e_coco \
--model_filename model.pdmodel \
--params_filename model.pdiparams \
--opset_version 11 \
--save_file output_inference/ppyoloe_crn_s_300e_coco/ppyoloe_crn_s_300e_coco.onnx
```

参数参考官方版本导出的参数设置

小技巧： 如果需要修改模型的输入尺寸，可以在上述导出的 onnx 模型进行修改，例如改为 384x640 的输入尺寸，使用以下命令进行修改:

```
python -m paddle2onnx.optimize \
--input_model ./output_inference/ppyoloe_crn_s_300e_coco/ppyoloe_crn_s_300e_coco.onnx \
--output_model ./output_inference/ppyoloe_crn_s_300e_coco/ppyoloe_384.onnx \
--input_shape_dict "{'x':[1,3,384,640]}"
```

8.3 onnx 模型转换 cvimodel

cvimodel 转换操作可以参考 cvimodel 转换操作可以参考 yolo-v5 移植章节的 onnx 模型转换 cvimodel 部分。

8.4 TDL_SDK 接口说明

预处理参数设置

预处理的设置接口如下所示

```
// set preprocess and algorithm param for ppyoloe detection
// if use official model, no need to change param
CVI_S32 init_param(const cvitdl_handle_t tdl_handle) {
  // setup preprocess
  YoloPreParam preprocess_cfg =
      CVI_TDL_Get_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_
↪PPYOLOE);

  for (int i = 0; i < 3; i++) {
    printf("asign val %d \n", i);
    preprocess_cfg.factor[i] = 0.003922;
    preprocess_cfg.mean[i] = 0.0;
  }
  preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("setup ppyoloe param \n");
  CVI_S32 ret =
      CVI_TDL_Set_YOLO_Preparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE,
↪ preprocess_cfg);
  if (ret != CVI_SUCCESS) {
    printf("Can not set ppyoloe preprocess parameters %#x\n", ret);
    return ret;
  }

  // setup yolo algorithm preprocess
  YoloAlgParam ppyoloe_param =
      CVI_TDL_Get_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_
↪PPYOLOE);
  ppyoloe_param.cls = 80;

  printf("setup ppyoloe algorithm param \n");
  ret = CVI_TDL_Set_YOLO_Algparam(tdl_handle, CVI_TDL_SUPPORTED_MODEL_
↪PPYOLOE, ppyoloe_param);
```

(续下页)

(接上页)

```
  if (ret != CVI_SUCCESS) {
    printf("Can not set ppyoloe algorithm parameters %#x\n", ret);
    return ret;
  }

  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, 0.5);
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, 0.
↪5);

  printf("ppyoloe algorithm parameters setup success!\n");
  return ret;
}
```

推理代码如下：

```
ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, model_
↪path.c_str());
if (ret != CVI_SUCCESS) {
  printf("open model failed %#x!\n", ret);
  return ret;
}
printf("cvimodel open success!\n");

std::cout << "model opened:" << model_path << std::endl;

VIDEO_FRAME_INFO_S fdFrame;
ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
std::cout << "CVI_TDL_ReadImage done!\n";

if (ret != CVI_SUCCESS) {
  std::cout << "Convert out video frame failed with :" << ret << ".file:" << str_src_dir
            << std::endl;
}

cvtdl_object_t obj_meta = {0};

CVI_TDL_PPYoloE(tdl_handle, &fdFrame, &obj_meta);

printf("detect number: %d\n", obj_meta.size);
for (uint32_t i = 0; i < obj_meta.size; i++) {
  printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_meta.info[i].bbox.y1,
         obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].bbox.score,
         obj_meta.info[i].classes);
}
```

8.5 测试结果

测试了 ppyoloe_crn_s_300e_coco 模型 onnx 以及 cvimodel 在 cv181x 平台的性能对比，其中阈值参数为：

- conf: 0.01
- nms: 0.7
- 输入分辨率：640 x 640

ppyoloe_crn_s_300e_coco 模型官方导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MI	MAP 0.5	MAP 0.5-0.95
pytorch	N/A	N/A	N/A	60.5	43.1
cv181x	103.62	110.59	14.68	量化失败	量化失败
cv182x	77.58	111.18	14.68	量化失败	量化失败
cv183x	量化失败	量化失败	量 化失败	量化失败	量化失败

ppyoloe_crn_s_300e_coco 模型的 TDL_SDK 导出方式 onnx 性能：

测试平台	推理耗时 (ms)	带宽 (MB)	ION(MB)	MAP 0.5	MAP 0.5-0.95
onnx	N/A	N/A	N/A	55.9497	39.8568
cv181x	101.15	103.8	14.55	55.36	39.1982
cv182x	75.03	104.95	14.55	55.36	39.1982
cv183x	30.96	80.43	13.8	55.36	39.1982s