Sophon Inference Documentation

Sophon

Jan 02, 2020

CONTENTS

注意

您购买的产品、服务或特性等应受比特大陆公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，比特大陆公司对本文档内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

技术支持

北京比特大陆科技有限公司

地址：北京市海淀区黑泉路宝胜南路 1 号院奥北科技园 25 号楼

邮编：100192

网址：www.sophon.ai

邮箱：support.ai@bitmain.com

发布记录

版本	发布时间	说明
V2.0.1	2019.11.15	第一次发布
V2.0.3	2020.01.01	增加示例程序

CHAPTER

ONE

快速入门

1.1 算丰硬件产品概览

我们提供了 4 种基于自研 AI 芯片的 TPU 产品。如果希望了解详细内容，可参考网站：https://sophon.ai

1.1.1 算丰加速卡 (SC) 系列

1.1.2 算丰边缘端小盒子 (SE) 系列

1.1.3 算丰服务器 (SA) 系列

1.1.4 算丰模组 (SM) 系列

敬请期待。

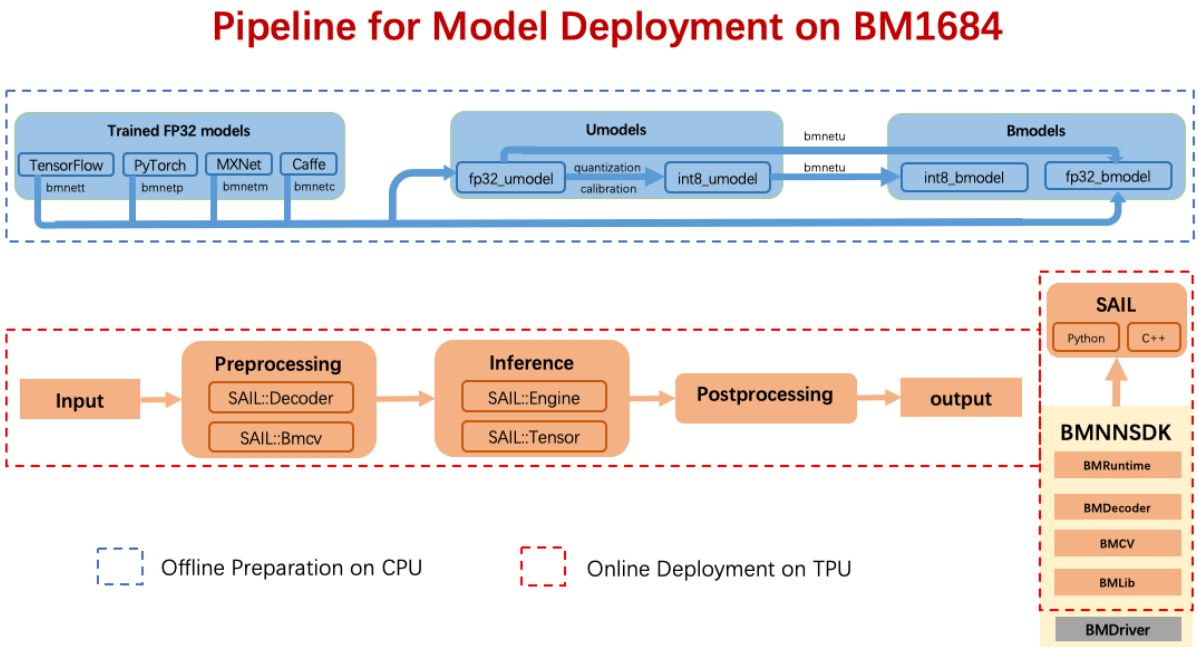
1.2 算丰软件产品概览

针对上一小节中提到的一系列算丰系列 TPU 产品，比特大陆自主研发了一套与之匹配的软件工具：Bitmain Neural Network Software Development Kit(BMNNSDK)。

使用算丰 TPU 产品所需的软件栈完全包含在 BMNNSDK 中。Sophon Inference 是 BMNNSDK 中的一个上层模块，提供了一系列的高级 API 以帮助用户快速部署模型。

在本小节中，我们首先总体讲述基于算丰 TPU 产品的深度学习模型部署的流程。然后，我们分别介绍 BMNNSDK 以及 Sophon Inference 的基本概念。最后，我们介绍 BMNNSDK 和 Sophon Inference 的安装使用以及相关注意事项。

1.2.1 模型部署



模型部署包含两步：模型的离线编译和在线推理。上图中包含的软件工具都包含在 BMNNSDK 中。

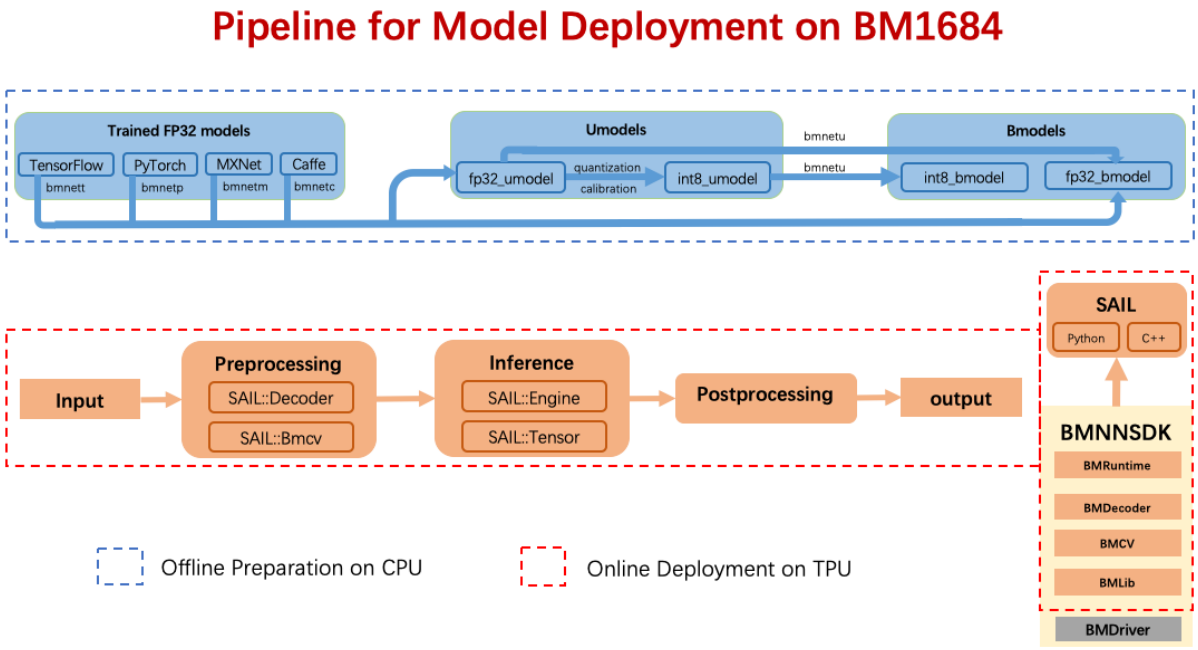
a). 模型离线编译

模型的离线编译对应上图中蓝色虚线框中的内容。这一过程的目的是把用户在各种框架下训练好的深度学习模型转换成我们定义的模型格式：bmodel。目前，算丰 TPU 包含了 FP32 和 INT8 两种类型的计算单元，因此 bmodel 也分为了 fp32_bmodel 和 int8_bmodel。如上图中，经过 tensorflow 训练生成的 xxx.pb 模型可以通过 bmnett 中提供的接口生成 fp32_bmodel；也可以先通过我们提供的脚本先生成 fp32_umodel，再通过量化校准工具生成 int8_umodel，最后通过 bmnetu 生成 int8_bmodel。目前，我们已经支持了将 Tensorflow、Pytorch、Mxnet、Caffe 四种深度学习框架下训练生成的模型编译成 bmodel。这一步骤的完成无需 TPU 的参与，因此是离线的。

b). 模型在线部署

模型的在线部署对应了上图中红色虚线框中的内容。bmodel 实际上是一系列算丰 TPU 指令的集合。通过使用我们提供的一系列运行时的接口，我们可以把 bmodel 中的指令加载到 TPU 上并执行。将 bmodel 加载到 TPU 内存上这一过程类似于将深度学习模型例如 xxx.pb 加载到主机内存的过程。加载 bmodel 完成之后，使用运行时的接口将输入张量发送给 TPU 并取出计算后得到的输出张量即是模型的推理过程。

1.2.2 BMNNSDK



BMNNSDK 是比特大陆自研的软件包。上图中提及的所有软件模块都包含在 BMNNSDK 中，包括了 Quantization & Calibration Tool, BMCompiler, BMDriver, BMLib, BMDecoder, BMCV, BMRuntime, Sophon Inference.

Quantization & Calibration Tool：该模块可以将 FP32 精度的模型转换成 INT8 精度的模型

在线文档: https://sophon-ai-algo.github.io/calibration_tools-doc/

BMCompiler：目前包括了五种模型编译工具。其中，bmnett 可以将 tensorflow 下训练生成的模型编译成 fp32_bmodel。bmnetp 可以将 pytorch 下训练生成的模型编译成 fp32_bmodel。bmnetm 可以将 mxnet 下训练生成的模型编译成 fp32_bmodel。bmnetc 可以将 caffe 下训练生成的模型编译成 fp32_bmodel。bmnetu 可以将 Quantization & Calibration Tool 下生成的 int8_umodel 编译成 int8_bmodel。

在线文档: https://sophon-ai-algo.github.io/bmnnsdk-doc/

BMDriver：是算丰 TPU 的驱动程序，将会通过 insmod 的方式安装到系统内核中。

BMLib：提供了一些基础接口，用来控制 TPU 与主机的内存交互。

在线文档: https://sophon-ai-algo.github.io/bmlib_1684-doc/

BMDecoder：提供了一些应用接口，用来驱动 TPU 上的硬件单元进行图像和视频的编解码。

在线文档: https://sophon-ai-algo.github.io/bm_multimedia/

BMCV：提供了一些应用接口，用来驱动 TPU 上的硬件单元进行张量计算和图像处理。

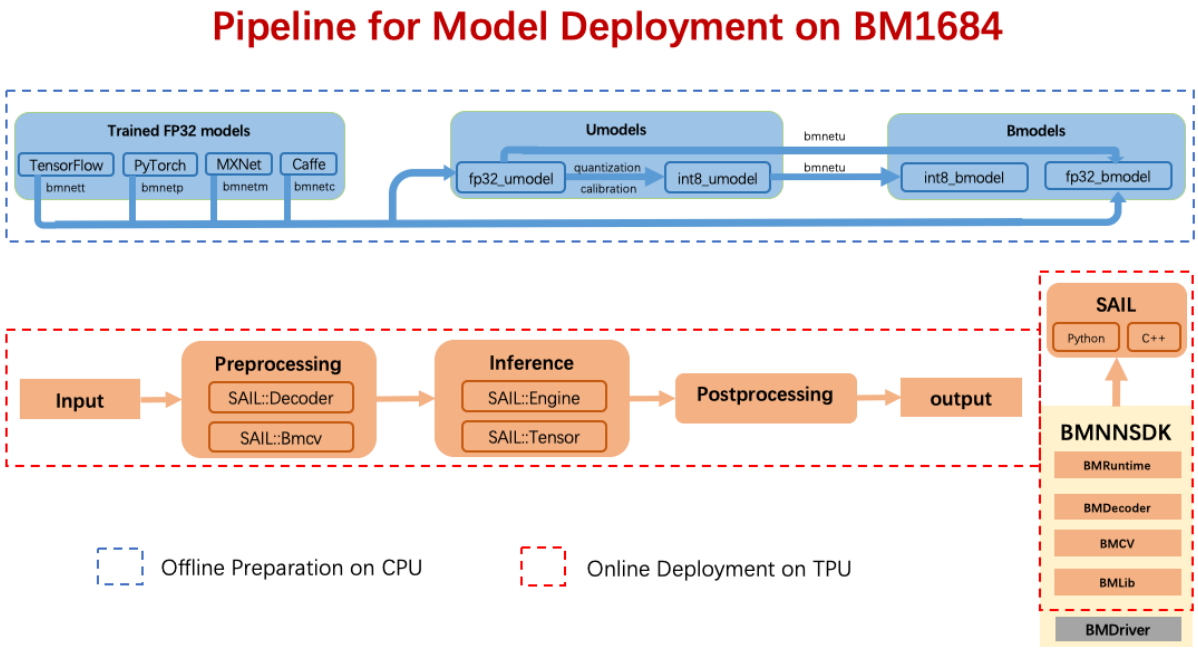
在线文档: https://sophon-ai-algo.github.io/bmcv_1684-doc/

BMRuntime：提供了一些应用接口，用来驱动 TPU 加载 bmodel 并进行模型推理。

在线文档: https://sophon-ai-algo.github.io/bmnnsdk-doc/

SAIL：提供了一些高级接口，主要是对 BMRuntime、BMCV、BMdecoder 等运行时模块的封装。

1.2.3 Sophon Inference



Sophon Inference 目前主要包含了 SAIL(Sophon Artificial Intelligent Library) 模块。我们提供了 python 和 c++ 的接口和示例程序，旨在帮助用户快速将模型部署到算丰 TPU 产品上。

SAIL：封装了 BMRuntime, BMCV, BMDecoder 等运行时模块。提供了 python 和 c++ 接口。可用来：

a). 驱动 TPU 加载 bmodel 并进行推理;

b). 驱动 TPU 进行图像和视频的处理。

在线英文文档: https://sophon-ai-algo.github.io/sophon-inference-doc_en/

在线中文文档: https://sophon-ai-algo.github.io/sophon-inference-doc_zh/

1.2.4 软件安装指南

在“1.1 算丰硬件产品概览”中，我们介绍了我们目前的四种产品形态：SC、SE、SA、SM。其中，SM 属于定制化的产品，因此在这里不做详细介绍。SC 系列产品为 PCIE 模式的加速卡，作为协处理器接受 X86 主机 CPU 的调用。SE 和 SA 系列产品为 SOC 模式，该模式下，操作系统运行在 TPU 内存上，由 TPU 上的 ARM 处理器负责管理和调度。

对于 SE 和 SA 系列产品的模型部署，我们通常在 X86 系统上编译模型生成 bmodel，再在 SE 和 SA 产品上部署。而在 SE 和 SA 系列产品上，我们已经预装了 BMNNSDK 中的全部运行时模块。因此，在这里我们只介绍 X86 主机下 PCIE 模式的 BMNNSDK 的安装。如果你希望你的模型最终运行在 SE 或 SA 产品上，那么只需了解 X86 主机下 BMNNSDK 中的离线模型编译工具的安装过程。

1.2.4.1 获取软件包并安装链接库

BMNNSDK 以 tar 包的形式发布。命名方式为 bmnnsdk2-bm1684_vx.x.x.tar.gz。其中，bmnnsdk2 代表版本号为 2，bm1684 代表支持的芯片编号，x.x.x 为详细版本号。解压该软件包后，我们用 ${BMNNSDK} 来代替软件包的主目录。

由于 BMNNSDK 中存在由不同版本内核编译生成的链接库，因此在解压完之后，我们需要根据当前主机的内核版本来选择适当的链接库。对此，我们提供了对应脚本。每次解压完之后只需运行一次下列命令即可。

```
cd ${BMNNSDK}/scripts/
./install_lib.sh nntc
```

1.2.4.2 离线模型编译工具安装

在”1.2.2 BMNNSDK“中我们介绍了 BMNNSDK 中的所有软件模块。离线模型编译工具包括了 Quantization & Calibration tool 和 BMCompiler。我们提供了一个脚本来完成离线工具的安装，每次进入终端之后运行以下命令即可完成安装。

```
cd ${BMNNSDK}/scripts/
source envsetup_pcie.sh
```

需要注意的是，由于 BMCompiler 依赖有比较多的依赖包，比如 bmnett 依赖 tensorflow，bmnetp 依赖 pytorch，bmnetm 依赖 mxnet。因此，如果你只需要其中某一个工具，可以带参数运行该脚本，如下：

```
cd ${BMNNSDK}/scrips/
# 安装 Quantization & Calibration Tool
source envsetup_pcie.sh ufw
# 安装 bmnetu
source envsetup_pcie.sh bmnetu
# 安装 bmnett
source envsetup_pcie.sh bmnett
# 安装 bmnetp
source envsetup_pcie.sh bmnetp
# 安装 bmnetm
source envsetup_pcie.sh bmnetm
# 安装 bmnetc
source envsetup_pcie.sh bmnetc
```

1.2.4.3 运行时工具安装

目前，需要安装的运行时工具只有 BMDriver 和 Sophon Inference 的 python 包。

安装 BMDriver 需要 root 权限，BMDriver 会在主机上编译并安装到系统内核中。

```
cd ${BMNNSDK}/scripts/
sudo ./install_driver_pcie.sh
```

安装 Sophon Inference：

```
cd ${BMNNSDK}/examples/sail/x86/
pip3 install --user sophon-x.x.x-py3-none-any.whl
```

1.3 小试牛刀：模型部署

在本章中，我们将帮助你快速地将一个由 tensorflow 训练生成的 mobilenet 部署到 Sophon SC5 加速卡上。在这之前，你需要准备一台安装了 Sophon SC5 加速卡的个人电脑，操作系统建议为 Ubuntu16.04。另外，你还需要一份 BMNNSDK 的软件包。

本教程包含三个步骤。首先，你需要安装驱动、bmnett、和 sophon-inference。这三个模块都在 BMNNSDK 软件包中。然后，我们使用 bmnett 工具将 mobilenet 转换成 bmodel。最后，我们使用 sophon-inference 把 bmodel 运行起来，对一张图片进行分类。

1.3.1 依赖软件安装

硬件环境确认

```
# 确认系统版本
  bitmain@bitmain:~$ lsb_release -a
  # No LSB modules are available.
  # Distributor ID: Ubuntu
  # Description:    Ubuntu 16.04.6 LTS
  # Release:        16.04
  # Codename:       xenial

# 确认内核版本
  bitmain@bitmain:~$ uname -r
  # 4.15.0-45-generic

# 确认 Sophon SC5 已正确插入 PCIE 插槽
  bitmain@bitmain:~$ lspci | grep 1684
  # 01:00.0 Processing accelerators: Device 1e30:1684 (rev 01)
```

解压并安装链接库

```
cd ${BMNNSDK}/scripts/
./install_lib.sh nntc
```

安装驱动

```
cd ${BMNNSDK}/scripts/
sudo ./install_driver_pcie.sh
```

安装 bmnett

```
cd ${BMNNSDK}/scripts/
source envsetup_pcie.sh bmnett
```

安装 Sophon Inference

```
cd ${BMNNSDK}/examples/sail/python3/x86/
pip3 install sophon-x.x.x-py3-none-any.whl --user
```

1.3.2 使用 bmnett 将 mobilenet 编译为 bmodel

我们已经在官网上传了一份 tensorflow mobilenet 的模型，直接下载即可：

```
wget https://sophon-file.bitmain.com.cn/sophon-prod/model/19/05/28/mobilenetv1_tf.
↪tar.gz
tar -zxvf mobilenetv1_tf.tar.gz
```

然后，使用下面的脚本将模型编译为 bmodel：

```
#!/usr/bin/env python3
import bmnett

model_path = "mobilenetv1.pb"  # path of tensorflow frozen model, which to be␣
↪converted.
outdir = "bmodel/"             # path of the generated bmodel.
target = "BM1684"              # targeted TPU platform, BM1684 or BM1682.
input_names = ["input"]        # input operation names.
output_names = ["MobilenetV1/Predictions/Reshape_1"]  # output operation names.
shapes = [(1, 224, 224, 3)]    # input shapes.
```

(continues on next page)

(continued from previous page)

```
net_name = "mobilenetv1"        # name of the generated bmodel.

bmnett.compile(model_path, outdir, target, input_names, output_names, shapes=shapes,␣
↪net_name=net_name)
```

运行结束之后，在上面脚本指定的 ${outdir} 目录下，会生成 compilation.bmodel.

1.3.3 使用 SAIL 驱动 TPU 加载 bmodel 并进行推理

```
#!/usr/bin/env python3

import cv2
import numpy as np
import sophon.sail as sail

bmodel = sail.Engine("bmodel/compilation.bmodel", 0, sail.IOMode.SYSIO)  #␣
↪initialize an Engine instance using bmodel.
graph_name = bmodel.get_graph_names()[0]                                 # graph_
↪name is just the net_name in conversion step.
input_tensor_name = bmodel.get_input_names(graph_name)[0]
# why transpose?
# bmodel will always be NCHW layout,
# so, if original tensorflow frozen model is formatted as NHWC,
# we should transpose original (1, 224, 224, 3) to (1, 3, 224, 224)
input_data = {input_tensor_name: np.transpose(np.expand_dims(cv2.resize(cv2.imread(
↪"cls.jpg"), (224,224)), 0), [0,3,1,2]).copy()}
outputs = bmodel.process(graph_name, input_data)                         # do␣
↪inference
```

CHAPTER

TWO

示例程序

2.1 前言

2.1.1 Demo 摘要

Binary	Input	Decoder	Preprocessor	Data Type	Model	Mode	Model Number	TPU Number	Batch Size	Multi-Thread
cls-resnet-0	image	opencv	opencv	fp32 int8	resnet-50	static	1	1	1	N
cls-resnet-1	image	opencv	opencv	fp32 int8	resnet-50	static	1	1	1	Y
cls-resnet-2	image	opencv	opencv	fp32 int8	resnet-50	static	1	2	1	Y
cls-resnet-3	image	opencv	opencv	fp32 int8	resnet-50	static	2	1	1	Y
det-ssd-0	video image	opencv	opencv	fp32 int8	ssd300-vgg16	static	1	1	1	N
det-ssd_1	video image	bm-ffmpeg	bmcv	fp32 int8	ssd300-vgg16	static	1	1	1	N
det-ssd-2	video image	bm-ffmpeg	bmcv	fp32 int8	ssd300-vgg16	static	1	1	4	N
det-ssd-3	video image	bm-opencv	bm-opencv	fp32 int8	ssd300-vgg16	static	1	1	1	N
det-ssd-4	video image	bm-opencv	bmcv	fp32 int8	ssd300-vgg16	static	1	1	1	N
det-yolov3-0	multi-video	opencv	opencv	fp32 int8	yolov3	static	1	1	1	Y
det-yolov3-1	multi-video	bm-ffmpeg	bmcv	fp32 int8	yolov3	static	1	1	1	Y
det-mtcnn	image	opencv	opencv	fp32	MTCNN	dynamic	1	1	1	N

如上表所示，我们准备了一系列示例程序以便让你更加快速地熟悉并使用 Sophon Inference。每个示例程序都提供了 c++ 和 python 的版本以供参考。示例程序分为 4 个类别，分别对应 4 种应用：

cls_resnet(基于 resnet50 的图像分类)

det_ssd(基于 ssd300-vgg16 的目标检测)

det_yolov3(基于 yolov3 的目标检测，多路视频)

det_mtcnn(基于 mtcnn 的人脸检测)

在每类示例程序中，我们也会使用不同的实现方案以应对各种具体的需求，上表中各示例程序的属性含义如下：

Binary: 可执行程序 (c++) 或脚本 (python) 的名称。

Input: 输入的数据类型，图像或者视频。

Decoder: 解码图像或者视频使用的依赖库，除 opencv 外，其余模块均调用 TPU 上的硬件单元进行计算。

Preprocessor: 图像预处理或张量计算的依赖库，除 opencv 外，其余模块均调用 TPU 上的硬件单元进行计算。

Data Type: bmodel 的数据类型，fp32 或 int8.

Model: 示例程序中使用的深度学习模型名称。

Mode: 动态或者静态模式，静态代表 bmodel 的输入张量尺寸不可变，动态代表可变。

Model Number: 示例程序同时加载的模型数量。

TPU Number: 示例程序同时使用的 TPU 数量。

Batch Size: 模型输入张量的 batch 大小。即，N 维度。

Multi Thread:: 示例程序使用的线程数量。

2.1.2 返回值

我们也定义了一个返回值列表，供调试参考。

ret	meaning
0	normal
1	comparing failed
2	invalid tpu id

2.2 使用 Resnet-50 进行图像分类

在本节，我们将展示如何在 Sophon TPU 上面使用 resnet-50 进行图像分类。我们使用的 bmodel 都是经过转换后的官方的 caffe 版本的 resnet-50，包括了 fp32 以及 int8 类型的。我们实现了 4 个示例程序。它们都是使用 opencv 进行图像解码和预处理。图像都会被缩放到固定的尺寸，在这里是 1*3*224*224。这 4 个程序的区别在于它们支持的模型数量、TPU 数量和线程数量不同。

ID	Input	Decoder	Preprocessor	Data Type	Model	Mode	Model Number	TPU Number	Multi-Thread
0	image	opencv	opencv	fp32 int8	resnet-50	static	1	1	N
1	image	opencv	opencv	fp32 int8	resnet-50	static	1	1	Y
2	image	opencv	opencv	fp32 int8	resnet-50	static	1	2	Y
3	image	opencv	opencv	fp32 int8	resnet-50	static	2	1	Y

2.2.1 运行 demo

2.2.1.1 获取 bmodel 和图片

在运行 demo 之前，我们需要下载 rensnet50 的 fp32_bmodel 和 int8_bmodel。另外，也需要一张图片作为待处理的输入。我们可以使用”dowload.py“下载。

```
python3 download.py resnet50_fp32.bmodel
python3 download.py resnet50_int8.bmodel
python3 download.py cls.jpg
```

2.2.1.2 运行 C++ 程序

For case 0:

```
# run fp32 bmodel
./cls_resnet_0 --bmodel ./resnet50_fp32.bmodel --input ./cls.jpg

# run int8 bmodel
./cls_resnet_0 --bmodel ./resnet50_fp32.bmodel --input ./cls.jpg
```

For case 1:

```
# run fp32 bmodel
./cls_resnet_1 --bmodel ./resnet50_fp32.bmodel --input ./cls.jpg --threads␣
↪2

# run int8 bmodel
./cls_resnet_1 --bmodel ./resnet50_int8.bmodel --input ./cls.jpg --threads␣
↪2
```

For case 2:

```
# run fp32 bmodel and int8 bmodel in two threads
./cls_resnet_2 --bmodel ./resnet50_fp32.bmodel --bmodel ./resnet50_int8.
↪bmodel --input ./cls.jpg
```

For case 3:

```
# run fp32 bmodel
./cls_resnet_3 --bmodel ./resnet50_fp32.bmodel --input ./cls.jpg --tpu_id␣
↪0 --tpu_id 1

# run int8 bmodel
./cls_resnet_3 --bmodel ./resnet50_int8.bmodel --input ./cls.jpg --tpu_id␣
↪0 --tpu_id 1
```

2.2.1.3 运行 python 程序

For case 0:

```
# run fp32 bmodel
python3 ./cls_resnet_0.py --bmodel ./resnet50_fp32.bmodel --input ./cls.
↪jpg --loops 1

# run int8 bmodel
python3 ./cls_resnet_0.py --bmodel ./resnet50_int8.bmodel --input ./cls.
↪jpg --loops 1
```

For case 1:

```
# run fp32 bmodel
python3 ./cls_resnet_1.py --bmodel ./resnet50_fp32.bmodel --input ./cls.
↪jpg --threads 2

# run int8 bmodel
python3 ./cls_resnet_1.py --bmodel ./resnet50_int8.bmodel --input ./cls.
↪jpg --threads 2
```

For case 2:

```
# run fp32 bmodel and int8 bmodel in two threads
python3 ./cls_resnet_2.py --bmodel ./resnet50_fp32.bmodel --bmodel ./
↪resnet50_int8.bmodel --input ./cls.jpg
```

For case 3:

```
# run fp32 bmodel
python3 ./cls_resnet_3.py --bmodel ./resnet50_fp32.bmodel --input ./cls.
↪jpg --tpu_id 0 --tpu_id 1

# run int8 bmodel
python3 ./cls_resnet_3.py --bmodel ./resnet50_int8.bmodel --input ./cls.
↪jpg --tpu_id 0 --tpu_id 1
```

2.2.2 C++ 代码解析

2.2.2.1 Case 0: 基础示例程序

在该示例中，我们封装了一个函数，完成图像分类的过程，如下：

```
bool inference(
const std::string& bmodel_path,
const std::string& input_path,
int                tpu_id,
int                loops,
const std::string& compare_path);
```

在该函数中，我们通过循环处理同一张图像来模拟真实的图像分类业务。

```
// pipeline of inference
for (int i = 0; i < loops; ++i) {
  // read image
  cv::Mat frame = cv::imread(input_path);
  // preprocess
  preprocessor.process(input, frame);
  // scale input data if input data type is int8 or uint8
  if (in_dtype != BM_FLOAT32) {
    engine.scale_input_tensor(graph_name, input_name, input);
  }
  // inference
  engine.process(graph_name);
  // scale output data if input data type is int8 or uint8
  if (out_dtype != BM_FLOAT32) {
    engine.scale_output_tensor(graph_name, output_name, output);
  }
  // postprocess
  auto result = postprocessor.process(output);
  // print result
```

(continues on next page)

(continued from previous page)

```
  // ...
}
```

如上代码所示，在每次循环中，我们首先使用 opencv 的” imread “函数解码图片。然后对图像进行预处理，获取 bmodel 的输入张量，并根据 bmodel 的数据类型缩放输入张量，缩放比例是 bmodel 中的参数，已经事先被加载到 engine 中。接着驱动 TPU 进行推理。最后对 TPU 输出的张量进行后处理，获得最终的结果 (top-5)。

在该示例程序中，图片解码和预处理的实现中直接调用了 opencv 的相关函数，同时后处理比较简单，只计算了 top-5 分类结果。因此你可以直接参考相关代码，下面我们主要介绍模型推理部分。

bmodel_path 是本例中用到的 resnet50 bmodel 的路径，fp32 或 int8 皆可。我们使用该 bmodel 初始化了一个 sail::Engine 的实例，该实例中保存了模型的基础信息，将作为我们后续模型推理的载体。如果你的机器上有多个 TPU，那么可以通过 tpu_id 指定使用哪个 TPU 进行推理，编号从 0 开始。

```
sail::Engine engine(bmodel_path, tpu_id, sail::SYSIO);
```

我们可以通过该实例提供的方法获取模型的属性，代码如下：

```
sail::Engine engine(bmodel_path, tpu_id, sail::SYSIO);
auto graph_name = engine.get_graph_names().front();
auto input_name = engine.get_input_names(graph_name).front();
auto output_name = engine.get_output_names(graph_name).front();
auto input_shape = engine.get_input_shape(graph_name, input_name);
auto output_shape = engine.get_output_shape(graph_name, output_name);
auto in_dtype = engine.get_input_dtype(graph_name, input_name);
auto out_dtype = engine.get_output_dtype(graph_name, output_name);
```

graph_name 代表 bmodel 中我们要使用的具体模型的名称，在该示例程序中，bmodel 中只包含一个模型。input_name 代表我们使用的具体模型中的输入张量名称，在该模型中，只有一个输入张量。output_name 代表我们使用的具体模型中的输出张量名称，在该模型中，只有一个输出张量。input_shape 和 output_shape 代表指定张量的尺寸。in_dtype 和 out_dtype 代表指定张量的数据类型。

实际上，你也可以使用 BMNNSDK 中的 bm_model.bin 工具获取上述信息，如下：

```
# fp32_bmodel
bitmain@bitmain:~$ bm_model.bin --info resnet50_fp32_191115.bmodel
# bmodel version: B.2.2
# chip: BM1684
# create time: Sat Nov 23 14:37:37 2019
#
# =========================================
# net: [ResNet-50_fp32]  index: [0]
# ------------
# stage: [0]  static
# input: data, [1, 3, 224, 224], float32
# output: fc1000, [1, 1000], float32

# int8_bmodel
# bitmain@bitmain:~$ bm_model.bin --info resnet50_int8_191115.bmodel
# bmodel version: B.2.2
# chip: BM1684
# create time: Sat Nov 23 14:38:50 2019
#
# =========================================
# net: [ResNet-50_int8]  index: [0]
# ------------
# stage: [0]  static
```

(continues on next page)

(continued from previous page)

```
# input: data, [1, 3, 224, 224], int8
# output: fc1000, [1, 1000], int8
```

2.2.2.2 Case 1: 单模型多线程

在 case 0 中，我们使用了 bmodel 初始化了一个 sail::Engine 的实例，模型推理的过程是在主线程中进行的。在 case 1 中，我们将展示如果在多个线程中使用同一个 sail::Engine 实例做模型推理。

我们定义了一个”thread_infer“函数来实现子线程中的模型推理逻辑，如下：

```
void thread_infer(
  int                 thread_id,
  sail::Engine*       engine,
  const std::string&  input_path,
  int                 loops,
  const std::string&  compare_path,
  std::promise<bool>& status);
```

在该函数中，我们也通过循环处理一张图片来模拟真实的图像分类业务，整体的代码逻辑与 case 0 类似，在这里不再重复介绍。

case 1 与 case 0 主要的区别在于对 sail::Engine 实例的处理上，下面简写为 engine。

首先，我们在主线程中创建 engine，使用了与 case 0 不同构造函数并使用 engine 的”load“函数加载 bmodel：

```
sail::Engine engine(tpu_id);

int ret = engine.load(bmodel_path);
```

不同于 case 0，使用该构造函数创建 engine 时，engine 中不会为 bmodel 的输入与输入张量创建内存，而是需要用户额外提供输入与输出张量，即 sail::Tensor 的实例。

因此，在子线程中，我们根据从 engine 实例中获取的模型信息创建对应张量：

```
// get handle to create input and output tensors
sail::Handle handle = engine->get_handle();
// allocate input and output tensors with both system and device memory
sail::Tensor in(handle, input_shape, in_dtype, true, true);
sail::Tensor out(handle, output_shape, out_dtype, true, true);
std::map<std::string, sail::Tensor*> input_tensors = {{input_name, &in}};
std::map<std::string, sail::Tensor*> output_tensors = {{output_name, &out}}
↪;
```

而在模型推理时，也需要指定对应的张量，选择下面的重载函数即可：

```
engine->process(graph_name, input_tensors, output_tensors);
```

2.2.2.3 Case 2: 多线程多模型

case 2 是 case 1 在模型数量上的扩展。在 case 1 中，我们在每个线程中都是用同一个在主线程中加载的 bmodel 做推理。而在 case 2 中，我们将会在 engine 中加载多个 bmodel，如下是加载模型的子线程函数：

```
/**
* @brief Load a bmodel.
*
* @param thread_id   Thread id
```

(continues on next page)

(continued from previous page)

```
* @param engine      Pointer to an Engine instance
* @param bmodel_path Path to bmodel
*/
void thread_load(
     int                thread_id,
     sail::Engine*      engine,
     const std::string& bmodel_path) {
       int ret = engine->load(bmodel_path);
       if (ret == 0) {
         auto graph_name = engine->get_graph_names().back();
         spdlog::info("Thread {} load {} successfully.", thread_id, graph_
↪name);
       }
     }
```

其它代码与 case 1 基本一致.

2.2.2.4 Case 3: 多线程多 TPU 模式

case 3 是 case 1 在 TPU 数量上的扩展。由于 engine 是与 TPU 一一对应的。因此我们将对每个线程创建一个指定 tpu_id 的 engine，如下：

```
// init Engine to load bmodel and allocate input and output tensors
// one engine for one TPU
std::vector<sail::Engine*> engines(thread_num, nullptr);
for (int i = 0; i < thread_num; ++i) {
  engines[i] = new sail::Engine(bmodel_path, tpu_ids[i], sail::SYSIO);
}
```

其它代码与 case 1 基本一致.

2.2.3 Python 代码解析

2.2.3.1 Case 0: 基础示例程序

在该示例程序中，我们封装了一个函数，完成图像分类的过程，如下：

```
def inference(bmodel_path, input_path, loops, tpu_id, compare_path):
""" Do inference of a model in a thread.

Args:
  bmodel_path: Path to bmodel
  input_path: Path to input image.
  loops: Number of loops to run
  compare_path: Path to correct result file
  status: Status of comparison

Returns:
  True for success and False for failure.
"""
```

在该函数中，我们通过循环处理同一张图像来模拟真实的图像分类业务。

```
# pipeline of inference
for i in range(loops):
  # read image and preprocess
  image = preprocess(input_path).astype(np.float32)
  # inference with fp32 input and output
```

(continues on next page)

(continued from previous page)

```
  # data scale(input: fp32 to int8, output: int8 to fp32) is done inside
  # for int8 model
  output = engine.process(graph_name, {input_name:image})
  # postprocess
  result = postprocess(output[output_name])
  # print result
  # ...
```

如上代码所示，在每次循环中，我们首先使用 opencv 的” imread “函数解码图片。然后对图像进行预处理，获取 bmodel 的输入张量。接着驱动 TPU 进行推理。最后对 TPU 输出的张量进行后处理，获得最终的结果 (top-5)。

在该示例程序中，图片解码和预处理的实现中直接调用了 opencv 的相关函数，同时后处理比较简单，只计算了 top-5 分类结果。因此你可以直接参考相关代码，下面我们主要介绍模型推理部分。

bmodel_path 是本例中用到的 resnet50 bmodel 的路径，fp32 或 int8 皆可。我们使用该 bmodel 初始化了一个 sail.Engine 的实例，该实例中保存了模型的基础信息，将作为我们后续模型推理的载体。如果你的机器上有多个 TPU，那么可以通过 tpu_id 指定使用哪个 TPU 进行推理，编号从 0 开始。

```
engine = sail.Engine(bmodel_path, tpu_id, sail.SYSIO)
```

我们可以通过该实例提供的方法获取模型的属性，代码如下：

```
graph_name = engine.get_graph_names()[0]
input_name = engine.get_input_names(graph_name)[0]
input_shape = engine.get_input_shape(graph_name, input_name)
output_name = engine.get_output_names(graph_name)[0]
output_shape = engine.get_output_shape(graph_name, output_name)
out_dtype = engine.get_output_dtype(graph_name, output_name);
```

graph_name 代表 bmodel 中我们要使用的具体模型的名称，在该示例程序中，bmodel 中只包含一个模型。input_name 代表我们使用的具体模型中的输入张量名称，在该模型中，只有一个输入张量。output_name 代表我们使用的具体模型中的输出张量名称，在该模型中，只有一个输出张量。input_shape 和 output_shape 代表指定张量的尺寸。in_dtype 和 out_dtype 代表指定张量的数据类型。

2.2.3.2 Case 1: 单模型多线程

在 case 0 中，我们使用了 bmodel 初始化了一个 sail.Engine 的实例，模型推理的过程是在主线程中进行的。在 case 1 中，我们将展示如果在多个线程中使用同一个 sail.Engine 实例做模型推理。

我们定义了一个” thread_infer “函数来实现子线程中的模型推理逻辑，如下：

```
def thread_infer(thread_id, engine, input_path, loops, compare_path,␣
↪status):
""" Do inference of a model in a thread.

Args:
  thread_id: ID of the thread
  engine: An sail.Engine instance
  input_path: Path to input image file
  loops: Number of loops to run
  compare_path: Path to correct result file
  status: Status of comparison

Returns:
```

(continues on next page)

(continued from previous page)

```
  None.
"""
```

在该函数中，我们也通过循环处理一张图片来模拟真实的图像分类业务，整体的代码逻辑与 case 0 类似，在这里不再重复介绍。

case 1 与 case 0 主要的区别在于对 sail.Engine 实例的处理上，下面简写为 engine。

首先，我们在主线程中创建 engine，使用了与 case 0 不同构造函数并使用 engine 的”load“函数加载 bmodel：

```
engine = sail.Engine(ARGS.tpu_id)
engine.load(ARGS.bmodel)
```

不同于 case 0，使用该构造函数创建 engine 时，engine 中不会为 bmodel 的输入与输入张量创建内存，而是需要用户额外提供输入与输出张量，即 sail.Tensor 的实例。

因此，在子线程中，我们根据从 engine 实例中获取的模型信息创建对应张量：

```
# get handle to create input and output tensors
handle = engine.get_handle()
input = sail.Tensor(handle, input_shape, in_dtype, True, True)
output = sail.Tensor(handle, output_shape, out_dtype, True, True)
input_tensors = {input_name:input}
ouptut_tensors = {output_name:output}
```

而在模型推理时，也需要指定对应的张量，选择下面的重载函数即可：

```
engine.process(graph_name, input_tensors, ouptut_tensors)
```

2.2.3.3 Case 2: 多线程多模型

case 2 是 case 1 在模型数量上的扩展。在 case 1 中，我们在每个线程中都是用同一个在主线程中加载的 bmodel 做推理。而在 case 2 中，我们将会在 engine 中加载多个 bmodel，如下是加载模型的子线程函数：

```
def thread_load(thread_id, engine, bmodel_path):
""" Load a model in a thread.

Args:
  thread_id: ID of the thread.
  engine: An sail.Engine instance.
  bmodel_path: Path to bmodel.

Returns:
  None.
"""
ret = engine.load(bmodel_path)
if ret == 0:
  graph_name = engine.get_graph_names()[-1]
  print("Thread {} load {} successfully.".format(thread_id, graph_name))
```

其它代码与 case 1 基本一致.

2.2.3.4 Case 3: 多线程多 TPU 模式

case 3 是 case 1 在 TPU 数量上的扩展。由于 engine 是与 TPU 一一对应的。因此我们将对每个线程创建一个指定 tpu_id 的 engine，如下：

```
# init Engine to load bmodel and allocate input and output tensors
# one engine for one TPU
engines = list()
thread_num = len(ARGS.tpu_id)
for i in range(thread_num):
  engines.append(sail.Engine(ARGS.bmodel, ARGS.tpu_id[i], sail.SYSIO))
```

其它代码与 case 1 基本一致.

2.3 使用 SSD 对图像和视频进行目标检测

在本节, 我们将展示如何在 Sophon TPU 上面使用 ssd300-vgg16 进行目标检测。我们使用的 bmodel 都是经过转换后的官方的 caffe 版本的 ssd300-vgg16，包括了 fp32 以及 int8 类型的。我们实现了 5 个示例程序。其中，case 0，case 1，case 3，case 4 的区别在于解码方式与数据预处理的方式不同。case 2 是 case 1 的多 btach 版本，可以体现 int8 模型在 batch 为 4 的倍数时较优异的性能。

在本节的示例程序中,我们将重点介绍图像/视频的解码和预处理。关于模型推理部分,即 sail::Engine 的使用，请参考“2.2 使用 Resnet-50 进行图像分类”中的相关说明。

ID	Input	Decoder	Prepro-cessor	Data Type	Model	Mode	Model Number	Batch Size	Multi-Thread
0	video image	opencv	opencv	fp32 int8	ssd300-vgg16	static	1	1	N
1	video image	bm-ffmpeg	bmcv	fp32 int8	ssd300-vgg16	static	1	1	N
2	video image	bm-ffmpeg	bmcv	fp32 int8	ssd300-vgg16	static	1	4	N
3	video image	bm-opencv	bm-opencv	fp32 int8	ssd300-vgg16	static	1	1	N
4	video image	bm-opencv	bmcv	fp32 int8	ssd300-vgg16	static	1	1	N

2.3.1 运行 demo

2.3.1.1 获取 bmodel、图像、视频

我们提供了”download.py“脚本下载所需的模型与数据。

```
python3 download.py ssd_fp32.bmodel
python3 download.py ssd_int8.bmodel
python3 download.py det.jpg
python3 download.py det.h264
```

2.3.1.2 运行 C++ 程序

For case 0:

```
# run fp32 bmodel with input of image
./det_ssd_0 --bmodel ./ssd_fp32.bmodel --input ./det.jpg --loops 1

# run int8 bmodel with input of video
./det_ssd_0 --bmodel ./ssd_int8.bmodel --input ./det.h264 --loops 1
```

For case 1:

```
# run fp32 bmodel with input of image
./det_ssd_1 --bmodel ./ssd_fp32.bmodel --input ./det.jpg --loops 1

# run int8 bmodel with input of video
./det_ssd_1 --bmodel ./ssd_int8.bmodel --input ./det.h264 --loops 1
```

For case 2:

```
# run int8 bmodel with input of video
./det_ssd_2 --bmodel ./ssd_int8.bmodel --input ./det.h264 --loops 1
```

For case 3:

```
# run fp32 bmodel with input of image
./det_ssd_3 --bmodel ./ssd_fp32.bmodel --input ./det.jpg --loops 1

# run int8 bmodel with input of video
./det_ssd_3 --bmodel ./ssd_int8.bmodel --input ./det.h264 --loops 1
```

For case 4:

```
# run fp32 bmodel with input of image
./det_ssd_4 --bmodel ./ssd_fp32.bmodel --input ./det.jpg --loops 1

# run int8 bmodel with input of video
./det_ssd_4 --bmodel ./ssd_int8.bmodel --input ./det.h264 --loops 1
```

2.3.1.3 运行 python 程序

For case 0:

```
# run fp32 bmodel with input of image
python3 ./det_ssd_0.py --bmodel ./ssd_fp32.bmodel --input ./det.jpg --
↪loops 1 --tpu_id 0 --compare verify_det_jpg_fp32_0.json

# run int8 bmodel with input of video
python3 ./det_ssd_0.py --bmodel ./ssd_int8.bmodel --input ./det.h264 --
↪loops 1 --tpu_id 0 --compare verify_det_h264_int8_0.json
```

For case 1:

```
# run fp32 bmodel with input of image
python3 ./det_ssd_1.py --bmodel ./ssd_fp32.bmodel --input ./det.jpg --
↪loops 1 --tpu_id 0 --compare verify_det_jpg_fp32_1.json

# run int8 bmodel with input of video
python3 ./det_ssd_1.py --bmodel ./ssd_int8.bmodel --input ./det.h264 --
↪loops 1 --tpu_id 0 --compare verify_det_h264_int8_0.json
```

For case 2:

```
# run int8 bmodel with input of video
python3 ./det_ssd_2.py --bmodel ./ssd_int8.bmodel --input ./det.h264 --
↪loops 1 --tpu_id 0 --compare verify_det_h264_int8_2.json
```

2.3.2 C++ 代码解析

2.3.2.1 Case 0: 使用 opencv 解码和数据预处理

在该示例中，我们封装了一个函数，完成目标检测的任务，如下：

```
/**
* @brief Load a bmodel and do inference.
*
* @param bmodel_path  Path to bmodel
* @param input_path   Path to input file
* @param tpu_id       ID of TPU to use
* @param loops        Number of loops to run
* @param compare_path Path to correct result file
* @return Program state
*     @retval true  Success
*     @retval false Failure
*/
bool inference(
     const std::string& bmodel_path,
     const std::string& input_path,
     int                tpu_id,
     int                loops,
     const std::string& compare_path);
```

在该函数中，我们通过循环处理同一张图像或者视频的连续帧来模拟真实的目标检测业务。

```
// pipeline of inference
for (int i = 0; i < loops; ++i) {
  // read an image from a image file or a video file
  cv::Mat frame;
  if (!decoder->read(frame)) {
    break;
  }
  // preprocess
  cv::Mat img1(input_shape[2], input_shape[3], is_fp32 ? CV_32FC3 : CV_
↪8SC3);
  preprocessor.process(frame, img1);
  mat_to_tensor_(img1, in);
  // inference
  engine.process(graph_name, input_tensors, input_shapes, output_tensors);
  auto real_output_shape = engine.get_output_shape(graph_name, output_
↪name);
  // postprocess
  float* output_data = reinterpret_cast<float*>(out.sys_data());
  std::vector<DetectRect> dets;
  postprocessor.process(dets, output_data, real_output_shape,
                        frame.cols, frame.rows);
  // ...
}
```

在该示例程序中，我们使用了 opencv 进行图像/视频的解码和预处理，CvDecoder、PreProcess 类中均封装了 opencv 的相关 API。

```
PreProcessor preprocessor(scale);

// ...

CvDecoder* decoder = CvDecoder::create(input_path);
```

2.3.2.2 Case 1: 使用 bm-ffmpeg 解码、使用 bmcv 做预处理

在该示例程序中，我们使用 bm-ffmpeg 做解码，bmcv 库做预处理。我们已经在 SAIL 封装了 bm-ffmpeg 与 bmcv 的 API，因此用户无需关注其底层实现。

如下，你可以将 sail::Decoder 看成是 cv::VideoCapture，将 sail::BMImage 看成 cv::Mat；

```
// init decoder.
// use bm-ffmpeg to decode video. default output format is compressed NV12
sail::Decoder decoder(input_path, true, tpu_id);
bool status = true;
// pipeline of inference
for (int i = 0; i < loops; ++i) {
  // read an image from a image file or a video file
  sail::BMImage img0 = decoder.read(handle);

  // do something...

}
```

2.3.2.3 Case 2: case 1 的 4N 模式

case 2 的流程与 case 1 几乎一致，但其 bmodel 的 batch 维度是 4。因此，需要 4 张图像或者 4 帧视频一起处理。

当 bmodel 的 batch 为 4 的倍数时，可以发挥出 TPU 上 int8 算力的最大性能。

2.3.2.4 Case 3: 使用 bm-opencv 进行解码和预处理

This case is suitale for SOC mode only. The form of calling bm-opencv in SOC mode is almost the same as calling opencv(public released) in PCIE mode.

2.3.2.5 Case 4: 使用 bm-opencv 解码、使用 bmcv 做预处理

This case is suitale for SOC mode only. The form of calling bm-opencv in SOC mode is almost the same as calling opencv(public released) in PCIE mode.

2.3.3 Python 代码解析

2.3.3.1 Case 0: 使用 opencv 解码和数据预处理

在该示例中，我们封装了一个函数，完成目标检测的任务，如下：

```
def inference(bmodel_path, input_path, loops, tpu_id, compare_path):
  """ Load a bmodel and do inference.
  Args:
    bmodel_path: Path to bmodel
    input_path: Path to input file
    loops: Number of loops to run
    tpu_id: ID of TPU to use
    compare_path: Path to correct result file

  Returns:
    True for success and False for failure
  """
```

在该函数中，我们通过循环处理视频的连续帧来模拟真实的目标检测业务。在本示例程序中，我们使用 opencv 进行视频解码和预处理。

```
# pipeline of inference
for i in range(loops):
  # read an image from a image file or a video file
  ret, img0 = cap.read()
  if not ret:
    break
  h, w, _ = img0.shape
  # preprocess
  data = preprocessor.process(img0)
  # inference
  input_tensors = {input_name: np.array([data], dtype=np.float32)}
  output = engine.process(graph_name, input_tensors)
  # postprocess
  dets = postprocessor.process(output[output_name], w, h)
  # print result
  # ...
```

2.3.3.2 Case 1: 使用 bm-ffmpeg 解码、使用 bmcv 做预处理

在该示例程序中，我们使用 bm-ffmpeg 做解码，bmcv 库做预处理。我们已经在 SAIL 封装了 bm-ffmpeg 与 bmcv 的 API，因此用户无需关注其底层实现。

```
# init decoder
decoder = sail.Decoder(input_path, True, tpu_id)
# pipeline of inference
for i in range(loops):
  # read an image from a image file or a video file
  img0 = decoder.read(handle)
  # do somethig ...
```

2.3.3.3 Case 2: case 1 的 4N 模式

The pipeline in case 2 is the same as that in case 1. But the batchsize in case 4 is 4. We want use this case to show you that, if you are using int8 computing units, batchsize is recommanded as 4 or multiples of 4. At this situation, you can use the TPU to its fullest.

2.3.3.4 Case 3: 使用 bm-opencv 做解码和数据预处理

This case is suitale for SOC mode only. The form of calling bm-opencv in SOC mode is almost the same as calling opencv(public released) in PCIE mode.

2.3.3.5 Case 4: 使用 bm-opencv 做解码，使用 bmcv 做预处理

This case is suitale for SOC mode only. The form of calling bm-opencv in SOC mode is almost the same as calling opencv(public released) in PCIE mode.

2.4 使用 Yolov3 对多路视频做目标检测

在本节，我们将展示如何在 Sophon TPU 上面使用 Yolov3 对多路视频中的物体进行检测。我们使用的 bmodel 都是经过转换后的官方的 darknet 版本的 Yolov3，包括了 fp32 以及 int8 类型的。我们实现了 2 个示例程序。它们分别使用 opencv 与 bm-ffmpeg 进行图像解码以及 opencv 与 bmcv 进行数据预处理。

ID	Input	Decoder	Prepro-cessor	Data Type	Model	Mode	Model Number	Batch Size	Multi-Thread
0	multi-video	opencv	opencv	fp32 int8	yolov3	static	1	1	Y
1	multi-video	bm-ffmpeg	bmcv	fp32 int8	yolov3	static	1	1	Y

2.4.1 运行 demo

2.4.1.1 获取 bmodel 和视频

运行”download.py“脚本可以下载所需的模型和数据。

```
python3 download.py yolov3_fp32.bmodel
python3 download.py yolov3_int8.bmodel
python3 download.py det.h264
```

2.4.1.2 运行 C++ 程序

For case 0:

```
./det_yolov3_0 --bmodel  ./yolov3_fp32.bmodel --input ./det.h264 --threads␣
↪2
./det_yolov3_0 --bmodel  ./yolov3_int8.bmodel --input ./det.h264 --threads␣
↪2
```

For case 1:

```
./det_yolov3_0 --bmodel  ./yolov3_fp32.bmodel --input ./det.h264 --threads␣
↪2
./det_yolov3_0 --bmodel  ./yolov3_int8.bmodel --input ./det.h264 --threads␣
↪2
```

2.4.1.3 运行 python 程序

For case 0:

```
python3 det_yolov3.py --bmodel ./yolov3_fp32.bmodel --input ./det.h264 --
↪loops 1 --tpu_id 1
python3 det_yolov3.py --bmodel ./yolov3_int8.bmodel --input ./det.h264 --
↪loops 1 --tpu_id 1
```

2.4.2 C++ 代码解析

2.4.2.1 Case 0: 使用 opencv 做解码和数据预处理

在该示例程序中，我们首先对 sail::Engine、PreProcessor、PostProcessor 初始化。其中，PreProcessor 中封装了 opencv 的 API。

```
// ...
sail::Engine engine(bmodel_path, tpu_id, sail::SYSIO);

// ...
```

(continues on next page)

(continued from previous page)

```
PreProcessor preprocessor(416, 416);

// ...

PostProcessor postprocessor(0.5);

// ...
```

然后，我们使用一个 while 循环来模拟真实的目标检测业务。

```
// ...

while (cap.read(frame)) {

  // ...

  preprocessor.processv2(input, frame);

  // ...

  engine.process(graph_name);

  // ...

  auto result = postprocessor.process(output, output_shape[2], height,␣
↪width);
```

2.4.2.2 Case 1: 使用 bm-ffmpeg 解码，使用 bmcv 做预处理

我们定义了 FFMpegFrameProvider 和 PreProcessorBmcv 类，它们分别封装了 sail::Decoder(bm-ffmpeg), sail::Bmcv(bmcv)。

```
// ...

PreProcessorBmcv preprocessor(bmcv, input_scale, 416, 416);
PostProcessor postprocessor(0.5);

// ...

FFMpegFrameProvider frame_provider(bmcv, input_path, tpu_id);
sail::BMImage img0, img1;

// ...

while (!frame_provider.get(img0)) {

  // ...

  preprocessor.process(img0, img1);

  // ...

  engine.process(graph_name);

  // ...

  auto result = postprocessor.process(output, output_shape[2], height,␣
↪width);
```

2.4.3 Python 代码解析

2.4.3.1 Case 0: 使用 opencv 做解码和数据预处理

如下，我们使用了一个 while 循环来模拟真实的目标检测业务：

```
# ...

while cap.isOpened():

  # ...

  ret, img = cap.read()

  # ...

  data = preprocess(img, detected_size)

  # ...

  input_data = {input_name: np.array([data], dtype=np.float32)}

  # ...

  output = net.process(graph_name, input_data)

  # ...

  bboxes, classes, probs = postprocess(output, img, detected_size,␣
↪threshold)

  # ...
```

2.5 使用 MTCNN 进行人脸检测

在本节中，我们使用 mtcnn 做人脸检测。

ID	Input	Decoder	Preprocessor	Data Type	Model	Mode	Model Number	TPU Number	Multi-Thread
0	image	opencv	opencv	fp32	MTCNN	dynamic	1	1	N

2.5.1 运行 demo

2.5.1.1 获取 bmodel 和图片

使用“download.py”下载所需的模型和数据.

```
python3 download.py mtcnn_fp32.bmodel
python3 download.py face.jpg
```

2.5.1.2 运行 C++ 程序

For case 0:

```
./det_mtcnn --bmodel ./mtcnn_fp32.bmodel --input ./face.jpg
```

2.5.1.3 运行 python 程序

For case 0:

```
python3 ./det_mtcnn.py --bmodel ./mtcnn_fp32.bmodel --input ./face.jpg
```

2.5.2 C++ 代码解析

2.5.2.1 Case 0

在本例中，我们使用的 bmodel 是一个动态模型，它的输入张量尺寸是可变的。bmodel 中有三个具体模型：PNet、RNet、ONet。其中，PNet 的输入张量的高和宽可变，RNet 和 ONet 的输入张量的 batch 可变。

```
// init Engine to load bmodel and allocate input and output tensors
sail::Engine engine(bmodel_path, tpu_id, sail::SYSIO);
// init preprocessor and postprocessor
PreProcessor preprocessor(127.5, 127.5, 127.5, 0.0078125);
double threshold[3] = {0.5, 0.3, 0.7};
PostProcessor postprocessor(threshold);
auto reference = postprocessor.get_reference(compare_path);
// read image
cv::Mat frame = cv::imread(input_path);
bool status = true;
for (int i = 0; i < loops; ++i) {
  cv::Mat image = frame.t();
  // run PNet, the first stage of MTCNN
  auto boxes = run_pnet(engine, preprocessor, postprocessor, image);
  if (boxes.size() != 0) {
    // run RNet, the second stage of MTCNN
    boxes = run_rnet(engine, preprocessor, postprocessor, boxes, image);
    if (boxes.size() != 0) {
      // run ONet, the third stage of MTCNN
      boxes = run_onet(engine, preprocessor, postprocessor, boxes, image);
    }
  }
  // print_result
  if (postprocessor.compare(reference, boxes)) {
    print_result(boxes);
  } else {
      status = false;
      break;
  }
}
```

2.5.3 Python 代码解析

2.5.3.1 Case 0

在本例中，我们使用的 bmodel 是一个动态模型，它的输入张量尺寸是可变的。bmodel 中有三个具体模型：PNet、RNet、ONet。其中，PNet 的输入张量的高和宽可变，RNet 和 ONet 的输入张量的 batch 可变。

```
# init Engine to load bmodel and allocate input and output tensors
engine = sail.Engine(bmodel_path, 0, sail.SYSIO)
# init preprocessor and postprocessor
preprocessor = PreProcessor([127.5, 127.5, 127.5], 0.0078125)
postprocessor = PostProcessor([0.5, 0.3, 0.7])
# read image
image = cv2.imread(input_path).astype(np.float32)
image = cv2.transpose(image)
# run PNet, the first stage of MTCNN
boxes = run_pnet(engine, preprocessor, postprocessor, image)
if np.array(boxes).shape[0] > 0:
  # run RNet, the second stage of MTCNN
  boxes = run_rnet(preprocessor, postprocessor, boxes, image)
  if np.array(boxes).shape[0] > 0:
    # run ONet, the third stage of MTCNN
    boxes, points = run_onet(preprocessor, postprocessor, boxes, image)
# print detected result
for i, bbox, prob in zip(range(len(boxes)), boxes, probs):
  print("Face {} Box: {}, Score: {}".format(i, bbox, prob))
```

CHAPTER

THREE

API 参考

3.1 SAIL

SAIL 是 Sophon Inference 中的核心模块，

SAIL 对 BMNNSDK 中的 BMLib、BMDecoder、BMCV、BMRuntime 进行了封装，将 BMNNSDK 中原有的 “加载 bmodel 并驱动 TPU 推理”、“驱动 TPU 做图像处理”、“驱动 VPU 做图像和视频解码” 等功能抽象成更为简单的 C++ 接口对外提供；并且支持使用 pybind11 再次封装，提供简洁的 python 接口。

目前，SAIL 模块中所有的类、枚举、函数都在 “sail” 名字空间下，本单元中的文档将深入介绍您可能用到的 SAIL 中的模块和类。核心的类包括：

- Handle：

BMNNSDK 中 BMLib 的 bm_handle_t 的包装类，设备句柄，上下文信息，用来和内核驱动交互信息。

- Tensor：

BMNNSDK 中 BMLib 的包装类，封装了对 device memroy 的管理以及与 system memory 的同步。

- Engine：

BMNNSDK 中 BMRuntime 的包装类，可以加载 bmodel 并驱动 TPU 进行推理。一个 Engine 实例可以加载一个任意的 bmodel，自动地管理输入张量与输出张量对应的内存。

- Decoder

使用 VPU 解码视频，JPU 解码图像，均为硬件解码。

- Bmcv：

BMNNSDK 中 BMCV 的包装类，封装了一系列的图像处理函数，可以驱动 TPU 进行图像处理。

3.2 SAIL C++ API

3.2.1 Basic function

1). get_available_tpu_num

```
/** @brief Get the number of available TPUs.
 *
 *  @return Number of available TPUs.
 */
int get_available_tpu_num();
```

3.2.2 Data type

1). bm_data_type_t

```
enum bm_data_type_t {
  BM_FLOAT32,     // float32
  BM_FLOAT16,     // not supported for now
  BM_INT8,        // int8
  BM_UINT8        // unsigned int8
};
```

3.2.3 Handle

1). Handle Constructor

```
/**
 * @brief Constructor using existed bm_handle_t.
 *
 * @param handle A bm_handle_t
 */
Handle(bm_handle_t handle);

/**
 * @brief Constructor with device id.
 *
 * @param dev_id Device id
 */
Handle(int dev_id);
```

2). data

```
/**
 *  @brief Get inner bm_handle_t.
 *
 *  @return Inner bm_handle_t
 */
bm_handle_t data();
```

3.2.4 Tensor

1). Tensor Constructor

```
/**
 * @brief Common constructor.
 * @detail
 *  case 0: only allocate system memory
 *          (handle, shape, dtype, true, false)
 *  case 1: only allocate device memory
 *          (handle, shape, dtype, false, true)
 *  case 2: allocate system memory and device memory
 *          (handle, shape, dtype, true, true)
 *
 * @param handle       Handle instance
 * @param shape        Shape of the tensor
 * @param own_sys_data Indicator of whether own system memory.
 * @param own_dev_data Indicator of whether own device memory.
 */
explicit Tensor(
    Handle                  handle,
```

(continues on next page)

(continued from previous page)

```
    const std::vector<int>& shape,
    bm_data_type_t          dtype,
    bool                    own_sys_data,
    bool                    own_dev_data);

/**
 *  @brief Copy constructor.
 *
 *  @param tensor A Tensor instance
 */
Tensor(const Tensor& tensor);
```

2). Tensor Assign Function

```
/**
 * @brief Assignment function.
 *
 * @param tensor A Tensor instance
 * @return A Tensor instance
 */
Tensor& operator=(const Tensor& tensor);
```

3). shape

```
/**
 * @brief Get shape of the tensor.
 *
 * @return Shape of the tensor
 */
const std::vector<int>& shape() const;
```

4). dtype

```
/**
 * @brief Get data type of the tensor.
 *
 * @return Data type of the tensor
 */
void dtype();
```

5). reshape

```
/**
 * @brief Reset shape of the tensor.
 *
 * @param shape Shape of the tensor
 */
void reshape(const std::vector<int>& shape);
```

6). own_sys_data

```
/**
 * @brief Judge if the tensor owns data in system memory.
 *
 * @return True for owns data in system memory.
 */
bool own_sys_data();
```

7). own_dev_data

```
/**
 * @brief Judge if the tensor owns data in device memory.
 *
 * @return True for owns data in device memory.
 */
bool own_dev_data();
```

8). sys_data

```
/**
 * @brief Get data pointer in system memory of the tensor.
 *
 * @return Data pointer in system memory of the tensor
 */
void* sys_data();
```

9). dev_data

```
/**
 * @brief Get pointer to device memory of the tensor.
 *
 * @return Pointer to device memory of the tensor
 */
bm_device_mem_t* dev_data();
```

10). reset_sys_data

```
/**
 * @brief Reset data pointer in system memory of the tensor.
 *
 * @param data  Data pointer in system memory of the tensor
 * @param shape Shape of the data
 */
void reset_sys_data(
    void*              data,
    std::vector<int>& shape);
```

11). reset_dev_data

```
/**
 * @brief Reset pointer to device memory of the tensor.
 *
 * @param data Pointer to device memory
 */
void reset_dev_data(bm_device_mem_t* data);
```

12). sync_s2d

```
/**
 * @brief Copy data from system memory to device memory.
 */
void sync_s2d();

/**
 * @brief Copy data from system memory to device memory with specified size.
 *
 * @param size Byte size to be copied
 */
void sync_s2d(int size);
```

13). sync_d2s

```
/**
 * @brief Copy data from device memory to system memory.
 */
void sync_d2s();

/**
 * @brief Copy data from device memory to system memory with specified size.
 *
 * @param size Byte size to be copied
 */
void sync_d2s(int size);
```

14). free

```
/**
 * @brief Free system and device memroy of the tensor.
 */
void free();
```

3.2.5 IOMode

1). IOMode

```
enum IOMode {
  /// Input tensors are in system memory while output tensors are
  /// in device memory.
  SYSI,
  /// Input tensors are in device memory while output tensors are
  /// in system memory.
  SYSO,
  /// Both input and output tensors are in system memory.
  SYSIO,
  /// Both input and output tensors are in device memory.
  DEVIO
};
```

3.2.6 Engine

1). Engine Constructor

```
/**
 * @brief Constructor does not load bmodel.
 *
 * @param tpu_id TPU ID. You can use bm-smi to see available IDs.
 */
 Engine(int tpu_id);

/**
 * @brief Constructor does not load bmodel.
 *
 * @param handle Handle created elsewhere.
 */
Engine(const Handle&   handle);

/**
 * @brief Constructor loads bmodel from file.
 *
 * @param bmodel_path Path to bmodel
```

(continues on next page)

(continued from previous page)

```
 * @param tpu_id      TPU ID. You can use bm-smi to see available IDs.
 * @param mode        Specify the input/output tensors are in system memory
 *                   or device memory
 */
Engine(
    const std::string& bmodel_path,
    int                tpu_id,
    IOMode             mode);

/**
 * @brief Constructor loads bmodel from file.
 *
 * @param bmodel_path Path to bmodel
 * @param handle      Handle created elsewhere.
 * @param mode        Specify the input/output tensors are in system memory
 *                    or device memory
 */
Engine(
    const std::string& bmodel_path,
    const Handle&      handle,
    IOMode             mode);

/**
 * @brief Constructor loads bmodel from system memory.
 *
 * @param bmodel_ptr  Pointer to bmodel in system memory
 * @param bmodel_size Byte size of bmodel in system memory
 * @param tpu_id      TPU ID. You can use bm-smi to see available IDs.
 * @param mode        Specify the input/output tensors are in system memory
 *                   or device memory
 */
 Engine(
     const void* bmodel_ptr,
     size_t      bmodel_size,
     int         tpu_id,
     IOMode      mode);

/**
 * @brief Constructor loads bmodel from system memory.
 *
 * @param bmodel_ptr  Pointer to bmodel in system memory
 * @param bmodel_size Byte size of bmodel in system memory
 * @param handle      Handle created elsewhere.
 * @param mode        Specify the input/output tensors are in system memory
 *                    or device memory
 */
Engine(
    const void*        bmodel_ptr,
    size_t             bmodel_size,
    const Handle&      handle,
    IOMode             mode);

/**
 * @brief Copy constructor.
 *
 * @param other An other Engine instance.
 */
Engine(const Engine& other);
```

2). Engine Assign Function

```
/**
 * @brief Assignment function.
 *
 * @param other An other Engine instance.
 * @return Reference of a Engine instance.
 */
Engine<Dtype>& operator=(const Engine& other);
```

3). get_handle

```
/**
 * @brief Get Handle instance.
 *
 * @return Handle instance
 */
Handle get_handle();
```

4). load

```
/**
 * @brief Load bmodel from file.
 *
 * @param bmodel_path Path to bmodel
 * @return Program state
 *     @retval true  Success
 *     @retval false Failure
 */
bool load(const std::string& bmodel_path);

/**
 * @brief Load bmodel from system memory.
 *
 * @param bmodel_ptr  Pointer to bmodel in system memory
 * @param bmodel_size Byte size of bmodel in system memory
 * @return Program state
 *     @retval true  Success
 *    @retval false Failure
 */
bool load(const void* bmodel_ptr, size_t bmodel_size);
```

5). get_graph_names

```
/**
 * @brief Get all graph names in the loaded bomodels.
 *
 * @return All graph names
 */
std::vector<std::string> get_graph_names();
```

6). set_io_mode

```
/**
 * @brief Set IOMode for a graph.
 *
 * @param graph_name The specified graph name
 * @param mode The specified IOMode
 */
void set_io_mode(
  const std::string& graph_name,
  IOMode             mode);
```

7). get_input_names

```
/**
 * @brief Get all input tensor names of the specified graph.
 *
 * @param graph_name The specified graph name
 * @return All the input tensor names of the graph
 */
std::vector<std::string> get_input_names(const std::string& graph_name);
```

8). get_output_names

```
/**
 * @brief Get all output tensor names of the specified graph.
 *
 * @param graph_name The specified graph name
 * @return All the output tensor names of the graph
 */
std::vector<std::string> get_output_names(const std::string& graph_name);
```

9). get_max_input_shapes

```
/**
 * @brief Get max shapes of input tensors in a graph.
 *
 * For static models, the max shape is fixed and it should not be changed.
 * For dynamic models, the tensor shape should be smaller than or equal to
 * the max shape.
 *
 * @param graph_name The specified graph name
 * @return Max shape of input tensors
 */
std::map<std::string, std::vector<int>> get_max_input_shapes(
    const std::string& graph_name);
```

10). get_input_shape

```
/**
 * @brief Get the shape of an input tensor in a graph.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return The shape of the tensor
 */
std::vector<int> get_input_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

11). get_max_output_shapes

```
/**
 * @brief Get max shapes of output tensors in a graph.
 *
 * For static models, the max shape is fixed and it should not be changed.
 * For dynamic models, the tensor shape should be smaller than or equal to
 * the max shape.
 *
 * @param graph_name The specified graph name
 * @return Max shape of output tensors
 */
std::map<std::string, std::vector<int>> get_max_output_shapes(
    const std::string& graph_name);
```

12). get_output_shape

```
/**
 * @brief Get the shape of an output tensor in a graph.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return The shape of the tensor
 */
std::vector<int> get_output_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

13). get_input_dtype

```
/**
 * @brief Get data type of an input tensor. Refer to bmdef.h as following.
 *   typedef enum {
 *     BM_FLOAT32 = 0,
 *     BM_FLOAT16 = 1,
 *     BM_INT8 = 2,
 *     BM_UINT8 = 3,
 *     BM_INT16 = 4,
 *     BM_UINT16 = 5,
 *     BM_INT32 = 6,
 *     BM_UINT32 = 7
 *   } bm_data_type_t;
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return Data type of the input tensor
 */
bm_data_type_t get_input_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

14). get_output_dtype

```
/**
 * @brief Get data type of an output tensor. Refer to bmdef.h as following.
 *   typedef enum {
 *     BM_FLOAT32 = 0,
 *     BM_FLOAT16 = 1,
 *     BM_INT8 = 2,
 *     BM_UINT8 = 3,
 *     BM_INT16 = 4,
 *     BM_UINT16 = 5,
 *     BM_INT32 = 6,
 *     BM_UINT32 = 7
 *   } bm_data_type_t;
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return Data type of the input tensor
 */
bm_data_type_t get_output_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

15). get_input_scale

```
/**
 * @brief Get scale of an input tensor. Only used for int8 models.
 *
```

(continues on next page)

(continued from previous page)

```
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return Scale of the input tensor
 */
float get_input_scale(
    const std::string& graph_name,
    const std::string& tensor_name);
```

16). get_output_scale

```
/**
 * @brief Get scale of an output tensor. Only used for int8 models.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return Scale of the output tensor
 */
float get_output_scale(
    const std::string& graph_name,
    const std::string& tensor_name);
```

17). reshape

```
/**
 * @brief Reshape input tensor for dynamic models.
 *
 * The input tensor shapes may change when running dynamic models.
 * New input shapes should be set before inference.
 *
 * @param graph_name   The specified graph name
 * @param input_shapes Specified shapes of all input tensors of the graph
 * @return 0 for success and 1 for failure
 */
int reshape(
    const std::string&                        graph_name,
    std::map<std::string, std::vector<int>>& input_shapes);
```

18). get_input_tensor

```
/**
 * @brief Get the specified input tensor.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return The specified input tensor
 */
Tensor* get_input_tensor(
    const std::string& graph_name,
    const std::string& tensor_name);
```

19). get_output_tensor

```
/**
 * @brief Get the specified output tensor.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @return The specified output tensor
 */
Tensor* get_output_tensor(
```

(continues on next page)

(continued from previous page)

```
    const std::string& graph_name,
    const std::string& tensor_name);
```

20). scale_input_tensor

```
/**
 * @brief Scale input tensor for int8 models.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @param data        Pointer to float data to be scaled
 */
void scale_input_tensor(
    const std::string& graph_name,
    const std::string& tensor_name,
    float*             data);
```

21). scale_output_tensor

```
/**
 * @brief Scale output tensor for int8 models.
 *
 * @param graph_name  The specified graph name
 * @param tensor_name The specified tensor name
 * @param data        Pointer to float data to be scaled
 */
void scale_output_tensor(
    const std::string& graph_name,
    const std::string& tensor_name,
    float*             data);
```

22). scale_fp32_to_int8

```
/**
 * @brief Scale data from float32 to int8. Only used for int8 models.
 *
 * @param src   Poniter to float32 data
 * @param dst   Poniter to int8 data
 * @param scale Value of scale
 * @param size  Size of data
 */
void scale_fp32_to_int8(float* src, int8_t* dst, float scale, int size);
```

23). scale_int8_to_fp32

```
/**
 * @brief Scale data from int8 to float32. Only used for int8 models.
 *
 * @param src   Poniter to int8 data
 * @param dst   Poniter to float32 data
 * @param scale Value of scale
 * @param size  Size of data
 */
void scale_int8_to_fp32(int8_t* src, float* dst, float scale, int size);
```

24). process

```
/**
 * @brief Inference with builtin input and output tensors.
 *
 * @param graph_name The specified graph name
```

(continues on next page)

(continued from previous page)

```
 */
void process(const std::string& graph_name);

/**
 * @brief Inference with provided input tensors.
 *
 * @param graph_name    The specified graph name
 * @param input_shapes  Shapes of all input tensors
 * @param input_tensors Data pointers of all input tensors in system memory
 */
void process(
    const std::string&                       graph_name,
    std::map<std::string, std::vector<int>>& input_shapes,
    std::map<std::string, void*>&            input_tensors);

/**
 * @brief Inference with provided input and output tensors.
 *
 * @param graph_name The specified graph name
 * @param input      Input tensors
 * @param output     Output tensors
 */
void process(
    const std::string&              graph_name,
    std::map<std::string, Tensor*>& input,
    std::map<std::string, Tensor*>& output);

/**
 * @brief Inference with provided input and output tensors and input shapes.
 *
 * @param graph_name   The specified graph name
 * @param input        Input tensors
 * @param input_shapes Real input tensor shapes
 * @param output       Output tensors
 */
void process(
    const std::string&                       graph_name,
    std::map<std::string, Tensor*>&          input,
    std::map<std::string, std::vector<int>>& input_shapes,
    std::map<std::string, Tensor*>&          output);
```

3.2.7 BMImage

1). BMImage Constructor

```
/**
 * @brief The default Constructor.
 */
BMImage();

/**
 * @brief The BMImage Constructor.
 *
 * @param handle A Handle instance
 * @param h      Image width
 * @param w      Image height
 * @param format Image format
 * @param dtype  Data type
 */
```

(continues on next page)

(continued from previous page)

```
BMImage(
    Handle&                  handle,
    int                      h,
    int                      w,
    bm_image_format_ext      format,
    bm_image_data_format_ext dtype);
```

2). data

```
/**
 * @brief Get inner bm_image
 *
 * @return The inner bm_image
 */
bm_image& data();
```

3). width

```
/**
 * @brief Get the img width.
 *
 * @return the width of img
 */
int width();
```

4). height

```
/**
 * @brief Get the img height.
 *
 * @return the height of img
 */
int height();
```

5). format

```
/**
 * @brief Get the img format.
 *
 * @return the format of img
 */
bm_image_format_ext format();
```

3.2.8 Decoder

1). Decoder Constructor

```
/**
 * @brief Constructor.
 *
 * @param file_path  Path or rtsp url to the video/image file.
 * @param compressed Whether the format of decoded output is compressed NV12.
 * @param tpu_id     ID of TPU, there may be more than one TPU for PCIE mode.
 */
Decoder(
    const std::string& file_path,
    bool               compressed = true,
    int                tpu_id = 0);
```

2). is_opened

```
/**
 * @brief Judge if the source is opened successfully.
 *
 * @return True if the source is opened successfully
 */
bool is_opened();
```

3). read

```
/**
 * @brief Read a bm_image from the Decoder.
 *
 * @param handle A bm_handle_t instance
 * @param image Reference of bm_image to be read to
 * @return 0 for success and 1 for failure
 */
int read(Handle& handle, bm_image& image);

/**
 * @brief Read a BMImage from the Decoder.
 *
 * @param handle A bm_handle_t instance
 * @param image Reference of BMImage to be read to
 * @return 0 for success and 1 for failure
 */
int read(Handle& handle, BMImage& image);
```

3.2.9 Bmcv

1). Bmcv Constructor

```
/**
 * @brief Constructor.
 *
 * @param handle A Handle instance
 */
explicit Bmcv(Handle handle);
```

2). bm_image_to_tensor

```
/**
 * @brief Convert BMImage to tensor.
 *
 * @param img    Input image
 * @param tensor Output tensor
 */
void bm_image_to_tensor(BMImage &img, Tensor &tensor);

/**
 * @brief Convert BMImage to tensor.
 *
 * @param img Input image
 */
Tensor bm_image_to_tensor(BMImage &img);
```

3). tensor_to_bm_image

```
/**
 * @brief Convert tensor to BMImage.
 *
```

(continues on next page)

(continued from previous page)

```
 * @param tensor   Input tensor
 * @param img      Output image
 */
void tensor_to_bm_image(Tensor &tensor, BMImage &img);

/**
 * @brief Convert tensor to BMImage.
 *
 * @param tensor   Input tensor
 */
BMImage tensor_to_bm_image(Tensor &tensor);
```

4). crop_and_resize

```
/**
 * @brief Crop then resize an image.
 *
 * @param input    Input image
 * @param output   Output image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @param resize_w Target width
 * @param resize_h Target height
 * @return 0 for success and other for failure
 */
int crop_and_resize(
    BMImage                     &input,
    BMImage                     &output,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h,
    int                         resize_w,
    int                         resize_h);

/**
 * @brief Crop then resize an image.
 *
 * @param input    Input image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @param resize_w Target width
 * @param resize_h Target height
 * @return Output image
 */
BMImage crop_and_resize(
    BMImage                     &input,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h,
    int                         resize_w,
    int                         resize_h);
```

5). crop

```
/**
```

(continues on next page)

(continued from previous page)

```
 * @brief Crop an image with given window.
 *
 * @param input    Input image
 * @param output   Output image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @return 0 for success and other for failure
 */
int crop(
    BMImage                     &input,
    BMImage                     &output,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h);

/**
 * @brief Crop an image with given window.
 *
 * @param input    Input image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @return Output image
 */
 BMImage crop(
    BMImage                     &input,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h);
```

6). resize

```
/**
 * @brief Resize an image with interpolation of INTER_NEAREST.
 *
 * @param input    Input image
 * @param output   Output image
 * @param resize_w Target width
 * @param resize_h Target height
 * @return 0 for success and other for failure
 */
int resize(
    BMImage                     &input,
    BMImage                     &output,
    int                         resize_w,
    int                         resize_h);

/**
 * @brief Resize an image with interpolation of INTER_NEAREST.
 *
 * @param input    Input image
 * @param resize_w Target width
 * @param resize_h Target height
 * @return Output image
 */
BMImage resize(
```

(continues on next page)

(continued from previous page)

```
    BMImage                     &input,
    int                         resize_w,
    int                         resize_h);
```

7). vpp_crop_and_resize

```
/**
 * @brief Crop then resize an image using vpp.
 *
 * @param input    Input image
 * @param output   Output image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @param resize_w Target width
 * @param resize_h Target height
 * @return 0 for success and other for failure
 */
int vpp_crop_and_resize(
    BMImage                     &input,
    BMImage                     &output,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h,
    int                         resize_w,
    int                         resize_h);

/**
 * @brief Crop then resize an image using vpp.
 *
 * @param input    Input image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @param resize_w Target width
 * @param resize_h Target height
 * @return Output image
 */
BMImage vpp_crop_and_resize(
    BMImage                     &input,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h,
    int                         resize_w,
    int                         resize_h);
```

8). vpp_crop

```
/**
 * @brief Crop an image with given window using vpp.
 *
 * @param input    Input image
 * @param output   Output image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
```

(continues on next page)

(continued from previous page)

```
 * @return 0 for success and other for failure
 */
int vpp_crop(
    BMImage                     &input,
    BMImage                     &output,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h);

/**
 * @brief Crop an image with given window using vpp.
 *
 * @param input    Input image
 * @param crop_x0  Start point x of the crop window
 * @param crop_y0  Start point y of the crop window
 * @param crop_w   Width of the crop window
 * @param crop_h   Height of the crop window
 * @return Output image
 */
BMImage vpp_crop(
    BMImage                     &input,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h);
```

9). vpp_resize

```
/**
 * @brief Resize an image with interpolation of INTER_NEAREST using vpp.
 *
 * @param input    Input image
 * @param output   Output image
 * @param resize_w Target width
 * @param resize_h Target height
 * @return 0 for success and other for failure
 */
 int vpp_resize(
     BMImage                     &input,
     BMImage                     &output,
     int                         resize_w,
     int                         resize_h);

/**
 * @brief Resize an image with interpolation of INTER_NEAREST using vpp.
 *
 * @param input    Input image
 * @param resize_w Target width
 * @param resize_h Target height
 * @return Output image
 */
BMImage vpp_resize(
    BMImage                     &input,
    int                         resize_w,
    int                         resize_h);
```

10). warp

```
/**
 * @brief Applies an affine transformation to an image.
```

(continues on next page)

(continued from previous page)

```
 *
 * @param input    Input image
 * @param output   Output image
 * @param matrix   2x3 transformation matrix
 * @return 0 for success and other for failure
 */
int warp(
    BMImage                             &input,
    BMImage                             &output,
    const std::pair<
      std::tuple<float, float, float>,
      std::tuple<float, float, float>> &matrix);

/**
 * @brief Applies an affine transformation to an image.
 *
 * @param input    Input image
 * @param matrix   2x3 transformation matrix
 * @return Output image
 */
BMImage warp(
    BMImage                             &input,
    const std::pair<
      std::tuple<float, float, float>,
      std::tuple<float, float, float>> &matrix);
```

11). convert_to

```
/**
 * @brief Applies a linear transformation to an image.
 *
 * @param input        Input image
 * @param output       Output image
 * @param alpha_beta   (a0, b0), (a1, b1), (a2, b2) factors
 * @return 0 for success and other for failure
 */
int convert_to(
    BMImage                       &input,
    BMImage                       &output,
    const std::tuple<
      std::pair<float, float>,
      std::pair<float, float>,
      std::pair<float, float>>   &alpha_beta);

/**
 * @brief Applies a linear transformation to an image.
 *
 * @param input        Input image
 * @param alpha_beta   (a0, b0), (a1, b1), (a2, b2) factors
 * @return Output image
 */
BMImage convert_to(
    BMImage                       &input,
    const std::tuple<
      std::pair<float, float>,
      std::pair<float, float>,
      std::pair<float, float>>   &alpha_beta);
```

12). yuv2bgr

```
/**
```

(continues on next page)

(continued from previous page)

```
 * @brief Convert an image from YUV to BGR.
 *
 * @param input    Input image
 * @param output   Output image
 * @return 0 for success and other for failure
 */
int yuv2bgr(
    BMImage                      &input,
    BMImage                      &output);

/**
 * @brief Convert an image from YUV to BGR.
 *
 * @param input    Input image
 * @return Output image
 */
BMImage yuv2bgr(BMImage  &input);
```

13). vpp_convert

```
/**
 * @brief Convert an image to BGR PLANAR format using vpp.
 *
 * @param input    Input image
 * @param output   Output image
 * @return 0 for success and other for failure
 */
int vpp_convert(
    BMImage  &input,
    BMImage  &output);

/**
 * @brief Convert an image to BGR PLANAR format using vpp.
 *
 * @param input    Input image
 * @return Output image
 */
BMImage vpp_convert(BMImage  &input);
```

14). convert

```
/**
 * @brief Convert an image to BGR PLANAR format.
 *
 * @param input    Input image
 * @param output   Output image
 * @return 0 for success and other for failure
 */
int convert(
    BMImage  &input,
    BMImage  &output);

/**
 * @brief Convert an image to BGR PLANAR format.
 *
 * @param input    Input image
 * @return Output image
 */
BMImage convert(BMImage  &input);
```

15). rectangle

```
/**
 * @brief Draw a rectangle on input image.
 *
 * @param image      Input image
 * @param x0         Start point x of rectangle
 * @param y0         Start point y of rectangle
 * @param w          Width of rectangle
 * @param h          Height of rectangle
 * @param color      Color of rectangle
 * @param thickness  Thickness of rectangle
 * @return 0 for success and other for failure
 */
int rectangle(
    BMImage                          &image,
    int                              x0,
    int                              y0,
    int                              w,
    int                              h,
    const std::tuple<int, int, int> &color,
    int                              thickness=1);
```

16). imwrite

```
/**
 * @brief Save the image to the specified file.
 *
 * @param filename   Name of the file
 * @param image      Image to be saved
 * @return 0 for success and other for failure
 */
int imwrite(
    const std::string &filename,
    BMImage           &image);
```

17). get_handle

```
/**
 * @brief Get Handle instance.
 *
 * @return Handle instance
 */
Handle get_handle();
```

3.3 SAIL Python API

SAIL use "pybind11" to wrap python interfaces, support python3.5.

3.3.1 Basic function

```
def get_available_tpu_num():
    """ Get the number of available TPUs.

    Returns
    -------
    tpu_num : int
        Number of available TPUs
    """
```

3.3.2 Data type

```
# Data type for float32
sail.Dtype.BM_FLOAT32
# Data type for int8
sail.Dtype.BM_INT8
# Data type for uint8
sail.Dtype.BM_UINT8
```

3.3.3 sail.Handle

```
def __init__(tpu_id):
    """ Constructor handle instance

    Parameters
    ----------
    tpu_id : int
        create handle with tpu Id
    """

def free():
    """ free handle
    """
```

3.3.4 sail.IOMode

```
# Input tensors are in system memory while output tensors are in device memory
sail.IOMode.SYSI
# Input tensors are in device memory while output tensors are in system memory.
sail.IOMode.SYSO
# Both input and output tensors are in system memory.
sail.IOMode.SYSIO
# Both input and output tensors are in device memory.
sail.IOMode.DEVIO
```

3.3.5 sail.Tensor

1). Tensor

```
def __init__(handle, data):
    """ Constructor allocates device memory of the tensor.

    Parameters
    ----------
    handle : sail.Handle
        Handle instance
    array_data : numpy.array
        Tensor ndarray data, dtype can be np.float32, np.int8 or np.uint8
    """

def __init__(handle, shape, dtype, own_sys_data, own_dev_data):
    """ Constructor allocates system memory and device memory of the tensor.

    Parameters
    ----------
    handle : sail.Handle
```

(continues on next page)

(continued from previous page)

```
        Handle instance
    shape : tuple
        Tensor shape
    dytpe : sail.Dtype
        Data type
    own_sys_data : bool
        Indicator of whether own system memory
    own_dev_data : bool
        Indicator of whether own device memory
    """
```

2). shape

```
def shape():
    """ Get shape of the tensor.

    Returns
    -------
    tensor_shape : list
        Shape of the tensor
    """
```

3). asnumpy

```
def asnumpy():
    """ Get system data of the tensor.

    Returns
    -------
    data : numpy.array
        System data of the tensor, dtype can be np.float32, np.int8
        or np.uint8 with respective to the dtype of the tensor.
    """

def asnumpy(shape):
    """ Get system data of the tensor.

    Parameters
    ----------
    shape : tuple
        Tensor shape want to get

    Returns
    -------
    data : numpy.array
        System data of the tensor, dtype can be np.float32, np.int8
        or np.uint8 with respective to the dtype of the tensor.
    """
```

4). update_data

```
def update_data(data):
    """ Update system data of the tensor. The data size should not exceed
        the tensor size, and the tensor shape will not be changed.

    Parameters
    -------
    data : numpy.array
        Data.
    """
```

5). scale_from

```
def scale_from(data, scale):
    """ Scale data to tensor in system memory.

    Parameters
    -------
    data : numpy.array with dtype of float32
        Data.
    scale : float32
        Scale value.
    """
```

6). scale_to

```
def scale_from(scale):
    """ Scale tensor to data in system memory.

    Parameters
    -------
    scale : float32
        Scale value.

    Returns
    -------
    data : numpy.array with dtype of float32
        Data.
    """

def scale_from(scale, shape):
    """ Scale tensor to data in system memory.

    Parameters
    -------
    scale : float32
        Scale value.
    shape : tuple
        Tensor shape want to get

    Returns
    -------
    data : numpy.array with dtype of float32
        Data.
    """
```

7). dtype

```
def dtype():
    """ Get data type of the tensor.

    Returns
    -------
    dtype : sail.Dtype
        Data type of the tensor
    """
```

8). reshape

```
def reshape(shape):
    """ Reset shape of the tensor.

    Parameters
    -------
    shape : list
```

(continues on next page)

(continued from previous page)

```
        New shape of the tensor
    """
```

9). own_sys_data

```
def own_sys_data():
    """ Judge if the tensor owns data pointer in system memory.

    Returns
    -------
    judge_ret : bool
        True for owns data pointer in system memory.
    """
```

10). own_dev_data

```
def own_dev_data():
    """ Judge if the tensor owns data in device memory.

    Returns
    -------
    judge_ret : bool
        True for owns data in device memory.
    """
```

11). sync_s2d

```
def sync_s2d():
    """ Copy data from system memory to device memory.
    """

def sync_s2d(size):
    """ Copy data from system memory to device memory with specified size.

    Parameters
    ----------
    size : int
        Byte size to be copied
    """
```

12). sync_d2s

```
def sync_d2s():
    """ Copy data from device memory to system memory.
    """

def sync_d2s(size):
    """ Copy data from device memory to system memory with specified size.

    Parameters
    ----------
    size : int
        Byte size to be copied
    """
```

3.3.6 sail.Engine

1). Engine

```
def __init__(tpu_id):
    """ Constructor does not load bmodel.

    Parameters
    ----------
    tpu_id : int
        TPU ID. You can use bm-smi to see available IDs
    """

def __init__(handle):
    """ Constructor does not load bmodel.

    Parameters
    ----------
    hanle : Handle
       A Handle instance
    """

def __init__(bmodel_path, tpu_id, mode):
    """ Constructor loads bmodel from file.

    Parameters
    ----------
    bmodel_path : str
        Path to bmodel
    tpu_id : int
        TPU ID. You can use bm-smi to see available IDs
    mode : sail.IOMode
        Specify the input/output tensors are in system memory
        or device memory
    """

def __init__(bmodel_path, handle, mode):
    """ Constructor loads bmodel from file.

    Parameters
    ----------
    bmodel_path : str
        Path to bmodel
    hanle : Handle
       A Handle instance
    mode : sail.IOMode
        Specify the input/output tensors are in system memory
        or device memory
    """

def __init__(bmodel_bytes, bmodel_size, tpu_id, mode):
    """ Constructor using default input shapes with bmodel which
    loaded in memory

    Parameters
    ----------
    bmodel_bytes : bytes
        Bytes of  bmodel in system memory
    bmodel_size : int
        Bmodel byte size
    tpu_id : int
        TPU ID. You can use bm-smi to see available IDs
    mode : sail.IOMode
        Specify the input/output tensors are in system memory
        or device memory
    """
```

(continues on next page)

(continued from previous page)

```
def __init__(bmodel_bytes, bmodel_size, handle, mode):
    """ Constructor using default input shapes with bmodel which
    loaded in memory

    Parameters
    ----------
    bmodel_bytes : bytes
        Bytes of  bmodel in system memory
    bmodel_size : int
        Bmodel byte size
    hanle : Handle
       A Handle instance
    mode : sail.IOMode
        Specify the input/output tensors are in system memory
        or device memory
    """
```

2). get_handle

```
def get_handle():
    """ Get Handle instance.

    Returns
    -------
    handle: sail.Handle
       Handle instance
    """
```

3). load

```
def load(bmodel_path):
    """ Load bmodel from file.

    Parameters
    ----------
    bmodel_path : str
        Path to bmodel
    """

def load(bmodel_bytes, bmodel_size):
    """ Load bmodel from file.

    Parameters
    ----------
    bmodel_bytes : bytes
        Bytes of  bmodel in system memory
    bmodel_size : int
        Bmodel byte size
    """
```

4). get_graph_names

```
def get_graph_names():
    """ Get all graph names in the loaded bmodels.

    Returns
    -------
    graph_names : list
        Graph names list in loaded context
    """
```

5). set_io_mode

```
def set_io_mode(mode):
    """ Set IOMode for a graph.

    Parameters
    ----------
    mode : sail.IOMode
        Specified io mode
    """
```

6). get_input_names

```
def get_input_names(graph_name):
    """ Get all input tensor names of the specified graph.

    Parameters
    ----------
    graph_name : str
        Specified graph name

    Returns
    -------
    input_names : list
        All the input tensor names of the graph
    """
```

7). get_output_names

```
def get_output_names(graph_name):
    """ Get all output tensor names of the specified graph.

    Parameters
    ----------
    graph_name : str
        Specified graph name

    Returns
    -------
    input_names : list
        All the output tensor names of the graph
    """
```

8). get_max_input_shapes

```
def get_max_input_shapes(graph_name):
    """ Get max shapes of input tensors in a graph.
        For static models, the max shape is fixed and it should not be changed.
        For dynamic models, the tensor shape should be smaller than or equal to
        the max shape.

    Parameters
    ----------
    graph_name : str
        The specified graph name

    Returns
    -------
    max_shapes : dict {str : list}
        Max shape of the input tensors
    """
```

9). get_input_shape

```
def get_input_shape(graph_name, tensor_name):
    """ Get the shape of an input tensor in a graph.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    tensor_name : str
        The specified input tensor name

    Returns
    -------
    tensor_shape : list
        The shape of the tensor
    """
```

10). get_max_output_shapes

```
def get_max_output_shapes(graph_name):
    """ Get max shapes of input tensors in a graph.
        For static models, the max shape is fixed and it should not be changed.
        For dynamic models, the tensor shape should be smaller than or equal to
        the max shape.

    Parameters
    ----------
    graph_name : str
        The specified graph name

    Returns
    -------
    max_shapes : dict {str : list}
        Max shape of the output tensors
    """
```

11). get_output_shape

```
def get_output_shape(graph_name, tensor_name):
    """ Get the shape of an output tensor in a graph.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    tensor_name : str
        The specified output tensor name

    Returns
    -------
    tensor_shape : list
        The shape of the tensor
    """
```

12). get_input_dtype

```
def get_input_dtype(graph_name, tensor_name)
    """ Get scale of an input tensor. Only used for int8 models.

    Parameters
    ----------
    graph_name : str
        The specified graph name
```

(continues on next page)

(continued from previous page)

```
    tensor_name : str
        The specified output tensor name

    Returns
    -------
    scale: sail.Dtype
        Data type of the input tensor
    """
```

13). get_output_dtype

```
def get_output_dtype(graph_name, tensor_name)
    """ Get scale of an output tensor. Only used for int8 models.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    tensor_name : str
        The specified output tensor name

    Returns
    -------
    scale: sail.Dtype
        Data type of the output tensor
    """
```

14). get_input_scale

```
def get_input_scale(graph_name, tensor_name)
    """ Get scale of an input tensor. Only used for int8 models.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    tensor_name : str
        The specified output tensor name

    Returns
    -------
    scale: float32
        Scale of the input tensor
    """
```

15). get_output_scale

```
def get_output_scale(graph_name, tensor_name)
    """ Get scale of an output tensor. Only used for int8 models.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    tensor_name : str
        The specified output tensor name

    Returns
    -------
    scale: float32
        Scale of the output tensor
```

(continues on next page)

(continued from previous page)

```
    """
```

16). process

```
def process(graph_name, input_tensors):
    """ Inference with provided system data of input tensors.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    input_tensors : dict {str : numpy.array}
        Data of all input tensors in system memory

    Returns
    -------
    output_tensors : dict {str : numpy.array}
        Data of all output tensors in system memory
    """

def process(graph_name, input_tensors, output_tensors):
    """ Inference with provided input and output tensors.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    input_tensors : dict {str : sail.Tensor}
        Input tensors managed by user
    output_tensors : dict {str : sail.Tensor}
        Output tensors managed by user
    """

def process(graph_name, input_tensors, input_shapes, output_tensors):
    """ Inference with provided input tensors, input shapes and output tensors.

    Parameters
    ----------
    graph_name : str
        The specified graph name
    input_tensors : dict {str : sail.Tensor}
        Input tensors managed by user
    input_shapes : dict {str : list}
        Shapes of all input tensors
    output_tensors : dict {str : sail.Tensor}
        Output tensors managed by user
    """
```

3.3.7 sail.BMImage

1). BMImage

```
def __init__():
    """ Constructor.
    """
```

2). width

```
def width():
    """ Get the img width.

    Returns
    ----------
    width : int
       The width of img
    """
```

3). height

```
def height():
    """ Get the img height.

    Returns
    ----------
    height : int
       The height of img
    """
```

4). format

```
def format():
    """ Get the img format.

    Returns
    ----------
    format : bm_image_format_ext
       The format of img
    """
```

3.3.8 sail.Decoder

1). Decoder

```
def __init__(file_path, compressed=True, tpu_id=0):
    """ Constructor.

    Parameters
    ----------
    file_path : str
       Path or rtsp url to the video/image file
    compressed : bool, default: True
       Whether the format of decoded output is compressed NV12.
    tpu_id: int, default: 0
       ID of TPU, there may be more than one TPU for PCIE mode.
    """
```

2). is_opened

```
def is_opened():
    """ Judge if the source is opened successfully.

    Returns
    ----------
    judge_ret : bool
        True for success and False for failure
    """
```

3). read

```
def read(handle, image):
    """ Read an image from the Decoder.

    Parameters
    ----------
    handle : sail.Handle
        Handle instance
    image : sail.BMImage
        BMImage instance
    Returns
    ----------
    judge_ret : int
        0 for success and others for failure
    """
```

3.3.9 sail.Bmcv

1). Bmcv

```
def __init__(handle):
    """ Constructor.

    Parameters
    ----------
    handle : sail.Handle
        Handle instance
    ""
```

2). bm_image_to_tensor

```
def bm_image_to_tensor(image):
    """ Convert image to tensor.

    Parameters
    ----------
    image : sail.BMImage
        BMImage instance

    Returns
    -------
    tensor : sail.Tensor
        Tensor instance
    """
```

3). tensor_to_bm_image

```
def tensor_to_bm_image(tensor):
    """ Convert tensor to image.

    Parameters
    ----------
    tensor : sail.Tensor
        Tensor instance

    Returns
    -------
    image : sail.BMImage
        BMImage instance
    """
```

4). crop_and_resize

```
def crop_and_resize(input, crop_x0, crop_y0, crop_w, crop_h, resize_w, resize_h):
    """ Crop then resize an image.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    crop_x0 : int
        Start point x of the crop window
    crop_y0 : int
        Start point y of the crop window
    crop_w : int
        Width of the crop window
    crop_h : int
        Height of the crop window
    resize_w : int
        Target width
    resize_h : int
        Target height

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

5). crop

```
def crop(input, crop_x0, crop_y0, crop_w, crop_h):
    """ Crop an image with given window.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    crop_x0 : int
        Start point x of the crop window
    crop_y0 : int
        Start point y of the crop window
    crop_w : int
        Width of the crop window
    crop_h : int
        Height of the crop window

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

6). resize

```
def resize(input, resize_w, resize_h):
    """ Resize an image with interpolation of INTER_NEAREST.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    resize_w : int
        Target width
    resize_h : int
```

(continues on next page)

(continued from previous page)

```
        Target height

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

7). vpp_crop_and_resize

```
def vpp_crop_and_resize(input, crop_x0, crop_y0, crop_w, crop_h, resize_w, resize_h):
    """ Crop then resize an image using vpp.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    crop_x0 : int
        Start point x of the crop window
    crop_y0 : int
        Start point y of the crop window
    crop_w : int
        Width of the crop window
    crop_h : int
        Height of the crop window
    resize_w : int
        Target width
    resize_h : int
        Target height

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

8). vpp_crop

```
def vpp_crop(input, crop_x0, crop_y0, crop_w, crop_h):
    """ Crop an image with given window using vpp.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    crop_x0 : int
        Start point x of the crop window
    crop_y0 : int
        Start point y of the crop window
    crop_w : int
        Width of the crop window
    crop_h : int
        Height of the crop window

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

9). vpp_resize

```
def vpp_resize(input, resize_w, resize_h):
    """ Resize an image with interpolation of INTER_NEAREST using vpp.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    resize_w : int
        Target width
    resize_h : int
        Target height

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

10). warp

```
def warp(input, matrix):
    """ Applies an affine transformation to an image.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    matrix: 2d list
        2x3 transformation matrix

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

11). convert_to

```
def convert_to(input, alpha_beta):
    """ Applies a linear transformation to an image.

    Parameters
    ----------
    input : sail.BMImage
        Input image
    alpha_beta: tuple
        (a0, b0), (a1, b1), (a2, b2) factors

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

12). yuv2bgr

```
def yuv2bgr(input):
    """ Convert an image from YUV to BGR.

    Parameters
    ----------
    input : sail.BMImage
```

(continues on next page)

(continued from previous page)

```
        Input image

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

13). vpp_convert

```
def vpp_convert(input):
    """ Convert an image to BGR PLANAR format using vpp.

    Parameters
    ----------
    input : sail.BMImage
        Input image

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

14). convert

```
def convert(input):
    """ Convert an image to BGR PLANAR format.

    Parameters
    ----------
    input : sail.BMImage
        Input image

    Returns
    ----------
    output : sail.BMImage
        Output image
    """
```

15). rectangle

```
def rectangle(image, x0, y0, w, h, color, thickness=1):
    """ Draw a rectangle on input image.

    Parameters
    ----------
    image : sail.BMImage
        Input image
    x0 : int
        Start point x of rectangle
    y0 : int
        Start point y of rectangle
    w : int
        Width of rectangle
    h : int
        Height of rectangle
    color : tuple
        Color of rectangle
    thickness : int
        Thickness of rectangle
```

(continues on next page)

(continued from previous page)

```
    Returns
    ----------
    process_status : int
        0 for success and others for failure
    """
```

16). imwrite

```
def imwrite(file_name, image):
    """ Save the image to the specified file.

    Parameters
    ----------
    file_name : str
        Name of the file
    output : sail.BMImage
        Image to be saved

    Returns
    ----------
    process_status : int
        0 for success and others for failure
    """
```

17). get_handle

```
def get_handle():
    """ Get Handle instance.

    Returns
    -------
    handle: sail.Handle
       Handle instance
"""
```