SOPHON-SAIL 使用手册

发布 3.8.0

SOPHGO

2024 年 06 月 06 日

目录

CHAPTER 1

声明

法律声明

注意

您购买的产品、服务或特性等应受算能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，算能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

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

邮编 100094

网址 https://www.sophgo.com/

邮箱 sales@sophgo.com

电话 +86-10-57590723 +86-10-57590724

CHAPTER 2

SAIL

2.1 SAIL

SAIL (SOPHON Artificial Intelligent Library) 是 SOPHON-SAIL 中的核心模块。SAIL 对 SOPHONSDK 中的 BMLib、sophon-mw(对于 BM1688 或者 CV186AH 处理器, 名称更改为 sophon-media)、BMCV、BMRuntime 进行了封装, 将 SOPHONSDK 中原有的“加载 bmodel 并驱动智能视觉深度学习处理器推理”、“驱动智能视觉深度学习处理器做图像处理”、“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; 并且使用 pybind11 再次封装，提供简洁易用的 python 接口。

目前，SAIL 模块中所有的类、枚举、函数都在 “sail” 命名空间下，本单元中的文档将向您深入介绍可能用到的 SAIL 中的模块和类。核心的类包括:

- Handle:

SDK 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:

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

- Engine:

SDK 中 BMRuntime 的包装类，可以加载 bmodel 并驱动智能视觉深度学习处理器进行推理。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。

- Encoder:

使用 VPU 编码视频，JPU 和软件编码图像。视频编码支持 h264，h265 的本地视频和 rtsp/rtmp 流；像素格式支持 NV12 和 I420。jpeg 硬件编码和其他图片格式的软件编码。

- Bmcv：

SDK 中 BMCV 的包装类，封装了一系列的图像处理函数，可以驱动智能视觉深度学习处理器进行图像处理。

SAIL 发布记录

版本	发布日期	说明
V2.0.0	2019.09.20	第一次发布。
V2.0.1	2019.11.16	V2.0.1 版本发布。
V2.0.3	2020.05.07	V2.0.3 版本发布。
V2.2.0	2020.10.12	V2.2.0 版本发布。
V2.3.0	2021.01.11	V2.3.0 版本发布。
V2.3.1	2021.03.09	V2.3.1 版本发布。
V2.3.2	2021.04.01	V2.3.2 版本发布。
V2.4.0	2021.05.23	V2.4.0 版本发布。
V2.5.0	2021.09.02	V2.5.0 版本发布。
V2.6.0	2022.01.30	V2.6.0 版本修正后发布。
V2.7.0	2022.03.16	V2.7.0 版本发布, 20220531 发布补丁版本。
V3.0.0	2022.07.16	V3.0.0 版本发布。
V3.1.0	2022.11.01	V3.1.0 版本发布。
V3.2.0	2022.12.01	V3.2.0 版本发布。
V3.3.0	2023.01.01	V3.3.0 版本发布。
V3.4.0	2023.03.01	V3.4.0 版本发布。
V3.5.0	2023.05.01	V3.5.0 版本发布。
V3.6.0	2023.07.01	V3.6.0 版本发布。
V3.7.0	2023.10.01	V3.7.0 版本发布。
V3.8.0	2024.04.10	V3.8.0 版本发布。

V3.8.0 更新内容

- 添加获取底板温度接口 get_board_temp
- 添加获取处理器温度接口 get_chip_temp
- 添加获取 tpu 利用率接口 get_tpu_util
- 添加获取 vpu 利用率接口 get_vpu_util
- 添加获取 vpp 利用率接口 get_vpp_util
- 添加获取处理器内存使用率接口 get_dev_stat

- 添加获取日志等级接口 set_loglevel
- Tensor 添加 d2d 接口
- Tensor 添加支持偏移量的 d2s、s2 接口
- Tensor 添加对 FLOAT16 的支持
- Bmcv 添加 fillRectangle、putText、imencode、open_water、mosaic、gaussian_blur、transpose、Sobel 等接口的支持
- 添加 yolov8 后处理异步接口 algo_yolov8_post_1output_async
- 添加 yolov8 后处理 CPU 加速异步接口 algo_yolov8_post_cpu_opt_1output_async
- 添加 yolov5 后处理 CPU 加速异步接口 algo_yolov5_post_cpu_opt_async
- 添加 deepsort 同步处理接口 deepsort_tracker_controller
- 添加 deepsort 异步处理接口 deepsort_tracker_controller_async
- 添加 bytetrack 同步处理接口 bytetrack_tracker_controller
- 添加 sort 同步处理接口 sort_tracker_controller
- 添加 sort 异步处理接口 sort_tracker_controller_async
- 添加使用 tpu 加速 openpose 后处理接口 tpu_kernel_api_openpose_part_nms
- 添加对 Windows 的支持
- 添加对 RISCV 的支持

V3.7.0 更新内容

- 添加 yolox 后处理异步接口 algo_yolox_post
- 添加使用 h264 和 h265 解码裸流接口
- Tensor 添加 dump 数据接口
- 添加使用 yolov5 后处理优化接口 algo_yolov5_post_cpu_opt
- 添加绘制多边形框功能接口
- 添加 cv::Mat 转换到 BMImage 的接口
- 添加对 Python3.9，Python3.10，Python3.11 版本的适配

V3.6.0 更新内容

- Decoder 添加保存视频接口 dump。
- 基于 BM1684X 添加对单输出的 yolov5 模型后处理使用智能视觉深度学习处理器进行加速的接口 tpu_kernel_api_yolov5_out_without_decode。

- 添加 deepsort 跟踪接口: deepsort_tracker_controller。
- 添加 bytetrack 跟踪接口: bytetrack_tracker_controller。
- convert_format 接口添加可以指定图像格式。
- Bmcv 添加 convert_yuv420p_to_gray 接口。

V3.5.0 更新内容

- 添加视频及图片编码接口 Encoder。
- Handle 添加获取设备型号的接口 get_target。
- 基于 BM1684X 添加对三输出的 yolov5 模型后处理使用智能视觉深度学习处理器进行加速的接口 tpu_kernel_api_yolov5_detect_out。
- 添加了调用多线程推理框架的 Python 测试例程。

CHAPTER 3

编译安装指南

3.1 源码目录结构

源码的目录结构如下:

```
└─── SOPHON-SAIL
    ├─── 3rdparty
    │    ├─── json
    │    ├─── prebuild
    │    ├─── pybind11
    │    ├─── pybind11_new
    │    └─── spdlog
    ├─── cmake                  # Cmake Files
    │    ├─── BM168x_ARM_PCIE
    │    ├─── BM168x_LoongArch64
    │    ├─── BM168x_RISCV
    │    └─── BM168x_SOC
    ├─── docs                   # Documentation codes
    │    ├─── common
    │    ├─── source_common
    │    └─── source_zh
    ├─── include                # Includes
    ├─── python                 # Wheel codes
    │    ├─── arm_pcie
    │    ├─── loongarch64
    │    ├─── pcie
    │    ├─── riscv
    │    ├─── soc
    │    └─── windows
    ├─── python_wheels          # Python Wheels
```

(下页继续)

(续上页)

```
|    ├─── arm_pcie
|    ├─── loongarch
|    ├─── soc_BM1684_BM1684X
|    └─── soc_BM1688
├─── sample                # Sample files
|    ├─── cpp
|    └─── python
└─── src                   # Source codes
```

其中 3rdparty 主要包含了编译 sail 需要依赖的第三方的一些头文件; cmake 中是编译用到的一些 cmake 文件; include 是 sail 的一些头文件; python 文件夹内包含了以下各平台下面 python wheel 的打包代码及脚本; python_wheels 文件夹内是一些预编译出来的 wheel 包,arm_pcie、loongarch、soc 三个文件夹分别是对应的平台; sample 文件夹内是一些示例程序; src 文件夹下面是各接口的实现代码。

3.2 SAIL 的编译及安装

注意: BM1688 处理器仅支持 soc 相关编译选项,BM1684 和 BM1684X 处理器无此限制。

3.2.1 编译参数

- BUILD_TYPE : 编译的类型, 目前有 pcie、soc、arm_pcie、loongarch、windows 五种模式, pcie 是编译在 x86 主机上可用的 SAIL 包,soc 表示使用交叉编译的方式, 在 x86 主机上编译 soc 上可用的 SAIL 包,arm_pcie 表示使用交叉编译的方式, 在 x86 主机上编译插有 bm168x 卡的 arm 主机上可用的 SAIL 包,loongarch 表示使用交叉编译的方式, 在 x86 主机上编译插有 bm168x 卡的 LoongArch64 架构主机上可用的 SAIL 包, windows 表示编译插有 bm168x 卡的 windows 主机上可用的 SAIL 包。默认 pcie。
- ONLY_RUNTIME : 编译结果是否只包含运行时, 而不包含 bmcv,sophon-ffmpeg,sophon-opencv, 如果此编译选项为 ON, 则 SAIL 的编解码及 Bmcv 接口不可用, 只有推理接口可用。默认 OFF。
- INSTALL_PREFIX : 执行 make install 时的安装路径,pcie 模式下默认 “/opt/sophon” , 与 libsophon 的安装路径一致, 交叉编译模式下默认 “build_soc”。
- PYTHON_EXECUTABLE : 编译使用的 “python3” 的路径名称 (路径 + 名称), 默认使用当前系统中默认的 python3。
- CUSTOM_PY_LIBDIR : 编译使用的 python3 的动态库的路径 (只包含路径), 默认使用当前系统中默认 python3 的动态库目录。
- LIBSOPHON_BASIC_PATH : 交叉编译模式下,libsophon 的路径, 如果配置不正确则会编译失败。pcie 模式下面此编译选项不生效。
- FFMPEG_BASIC_PATH : 交叉编译模式下,sophon-ffmpeg 的路径, 如果配置不正确, 且 ONLY_RUNTIME 为 “OFF” 时会编译失败。pcie 模式下面此编译选项不生效。

- OPENCV_BASIC_PATH：交叉编译模式下,sophon-opencv 的路径, 如果配置不正确, 且 ONLY_RUNTIME 为 “OFF” 时会编译失败。pcie 模式下面此编译选项不生效。
- TOOLCHAIN_BASIC_PATH：交叉编译模式下, 交叉编译器的路径, 目前只有在 BUILD_TYPE 为 loongarch 时生效。
- BUILD_PYSAIL：编译结果是否包含 python 版 SAIL, 默认为为 “ON”，包含 python 版本 SAIL。
- TARGET_TYPE： windows 下的编译类型，当前支持 “release” 模式。
- RUNTIME_LIB： windows 下的库类型，当前支持 “MT” 模式。

3.2.2 编译可被 C++ 接口调用的动态库及头文件

注意: BM1688 和 CV186AH 处理器仅支持 SOC MODE 章节,BM1684 和 BM1684X 处理器无此限制。对于 BM1688 或者 CV186AH 处理器, 下列命令中的 sophon-mw 应改为 sophon-media。

PCIE MODE

. 安装 libsophon,sophon-ffmpeg,sophon-opencv 的 SAIL

libsophon,sophon-ffmpeg,sophon-opencv 的安装方式可参考算能官方文档

. 典型编译方式一

使用默认安装路径, 编译包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_PYSAIL=OFF ..
make sail
```

4. 安装 SAIL 动态库及头文件, 编译结果将安装在‘/opt/sophon’下面

```
sudo make install
```

. 典型编译方式二

使用默认安装路径, 编译不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL,

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DONLY_RUNTIME=ON -DBUILD_PYSAIL=OFF ..
make sail
```

4. 安装 SAIL 动态库及头文件, 编译结果将安装在‘/opt/sophon’下面

```
sudo make install
```

SOC MODE

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.4.1, sophon-ffmpeg 的版本为 0.4.1,sophon-opencv 的版本为 0.4.1。

1. 从算能官网中获取‘libsophon_soc_0.4.1_aarch64.tar.gz’, 并解压

```
tar -xvf libsophon_soc_0.4.1_aarch64.tar.gz
```

解压后 libsophon 的目录为‘libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1’

2. 从算能官网中获取‘sophon-mw-soc_0.4.1_aarch64.tar.gz’, 并解压

```
tar -xvf sophon-mw-soc_0.4.1_aarch64.tar.gz
```

解压后 sophon-ffmpeg 的目录为‘sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1’。

解压后 sophon-opencv 的目录为‘sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1’。

. 安装 g++-aarch64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

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

. 典型编译方式一

通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=soc -DBUILD_PYSAIL=OFF \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
  -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 \
  -DFFMPEG_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
↪sophon-ffmpeg_0.4.1 \
  -DOPENCV_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
↪sophon-opencv_0.4.1 ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_soc’, 编译结果将安装在‘build_soc’下面

```
make install
```

5. 将‘build_soc’文件夹下的‘sophon-sail’拷贝至目标 SOC 的‘/opt/sophon’目录下, 即可在 soc 上面进行调用。

. 典型编译方式二

通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=soc  \
  -DBUILD_PYSAIL=OFF \
  -DONLY_RUNTIME=ON \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
  -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_soc’, 编译结果将安装在‘build_soc’下面

```
make install
```

5. 将‘build_soc’文件夹下的‘sophon-sail’拷贝至目标 SOC 的‘/opt/sophon’目录下, 即可在 soc 上进行调用。

ARM PCIE MODE

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.4.1, sophon-ffmpeg 的版本为 0.4.1,sophon-opencv 的版本为 0.4.1。

1. 从算能官网中获取‘libsophon_0.4.1_aarch64.tar.gz’, 并解压

```
tar -xvf libsophon_0.4.1_aarch64.tar.gz
```

解压后 libsophon 的目录为‘libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1’

2. 从算能官网中获取‘sophon-mw_0.4.1_aarch64.tar.gz’, 并解压

```
tar -xvf sophon-mw_0.4.1_aarch64.tar.gz
```

解压后 sophon-ffmpeg 的目录为‘sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1’。

解压后 sophon-opencv 的目录为‘sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1’。

. 安装 g++-aarch64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

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

. 典型编译方式一

通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=arm_pcie  \
  -DBUILD_PYSAIL=OFF \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
↪ToolChain_aarch64_linux.cmake \
  -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 \
  -DFFMPEG_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
↪ffmpeg_0.4.1 \
  -DOPENCV_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
↪opencv_0.4.1 ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_arm_pcie’, 编译结果将安装在‘build_arm_pcie’下面

```
make install
```

5. 将‘build_arm_pcie’文件夹下的‘sophon-sail’拷贝至目标 ARM 主机的‘/opt/sophon’目录下, 即可在目标机器上面进行调用。

. 典型编译方式二

通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=arm_pcie  \
   -DONLY_RUNTIME=ON \
   -DBUILD_PYSAIL=OFF \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
↪ToolChain_aarch64_linux.cmake \
   -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_arm_pcie’, 编译结果将安装在‘build_arm_pcie’下面

```
make install
```

5. 将‘build_arm_pcie’文件夹下的‘sophon-sail’拷贝至目标 ARM 主机的‘/opt/sophon’目录下, 即可在目标机器上面进行调用。

LOONGARCH64 MODE

. 安装 loongarch64-linux-gnu 工具链

从 LoongArch64 官网获取其 [交叉编译的工具链](http://ftp.loongnix.cn/toolchain/gcc/release/loongarch/gcc8/loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1.tar.xz), 解压到本地, 解压后的目录结构如下:

```
└── loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1
    ├── bin
    ├── lib
    ├── lib64
```

(下页继续)

(续上页)

```
├──── libexec
├──── loongarch64-linux-gnu
├──── share
├──── sysroot
└──── versions
```

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.4.7, sophon-ffmpeg 的版本为 0.6.0,sophon-opencv 的版本为 0.6.0。

. 典型编译方式一

通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL,

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=loongarch  \
   -DBUILD_PYSAIL=OFF \
   -DTOOLCHAIN_BASIC_PATH=toolchains/loongson-gnu-toolchain-8.3-x86_
↪64-loongarch64-linux-gnu-rc1.1 \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_LoongArch64/
↪ToolChain_loongarch64_linux.cmake \
   -DLIBSOPHON_BASIC_PATH=libsophon_0.4.7_loongarch64/opt/sophon/
↪libsophon-0.4.7 \
   -DFFMPEG_BASIC_PATH=sophon-mw_0.6.0_loongarch64/opt/sophon/
↪sophon-ffmpeg_0.6.0 \
   -DOPENCV_BASIC_PATH=sophon-mw_0.6.0_loongarch64/opt/sophon/
↪sophon-opencv_0.6.0  \
   ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_loongarch’, 编译结果将安装在‘build_loongarch’下面

```
make install
```

5. 将‘build_loongarch’文件夹下的‘sophon-sail’拷贝至目标龙芯主机的‘/opt/sophon’目录下, 即可在目标机器上调用。

. 典型编译方式二

通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录

2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=loongarch  \
   -DBUILD_PYSAIL=OFF \
   -DONLY_RUNTIME=ON \
   -DTOOLCHAIN_BASIC_PATH=toolchains/loongson-gnu-toolchain-8.3-x86_
↪64-loongarch64-linux-gnu-rc1.1 \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_LoongArch64/
↪ToolChain_loongarch64_linux.cmake \
   -DLIBSOPHON_BASIC_PATH=libsophon_0.4.7_loongarch64/opt/sophon/
↪libsophon-0.4.7 \
   ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_loongarch’, 编译结果将安装在‘build_loongarch’下面

```
make install
```

5. 将‘build_loongarch’文件夹下的‘sophon-sail’拷贝至目标龙芯主机的‘/opt/sophon’目录下, 即可在目标机器上调用。

RISCV MODE

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.5.0, sophon-ffmpeg 的版本为 0.6.0,sophon-opencv 的版本为 0.6.0。

1. 从算能官网中获取‘libsophon_0.5.0_riscv64.tar.gz’, 并解压

```
tar -xvf libsophon_0.4.0_riscv64.tar.gz
```

解压后 libsophon 的目录为‘libsophon_0.5.0_riscv64/opt/sophon/libsophon-0.5.0’

2. 从算能官网中获取‘sophon-mw_0.6.0_riscv_64.tar.gz’, 并解压

```
tar -xvf sophon-mw_0.6.0_riscv_64.tar.gz
```

解压后 sophon-ffmpeg 的目录为‘sophon-mw_0.6.0_riscv_64/opt/sophon/sophon-ffmpeg_0.6.0’。

解压后 sophon-opencv 的目录为‘sophon-mw_0.6.0_riscv_64/opt/sophon/sophon-opencv_0.6.0’。

. 安装 g++-riscv64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

```
sudo apt-get install gcc-riscv64-linux-gnu g++-riscv64-linux-gnu
```

. 典型编译方式一

通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=riscv  \
  -DBUILD_PYSAIL=OFF \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_RISCV/ToolChain_
↪riscv64_linux.cmake \
  -DLIBSOPHON_BASIC_PATH=libsophon_0.5.0_riscv64/opt/sophon/
↪libsophon-0.5.0 \
  -DFFMPEG_BASIC_PATH=sophon-mw_0.6.0_riscv_64/opt/sophon/
↪sophon-ffmpeg_0.6.0 \
  -DOPENCV_BASIC_PATH=sophon-mw_0.6.0_riscv_64/opt/sophon/
↪sophon-opencv_0.6.0 ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_riscv’, 编译结果将安装在‘build_riscv’下面

```
make install
```

5. 将‘build_riscv’文件夹下的‘sophon-sail’拷贝至目标 RISCV 主机的‘/opt/sophon’目录下, 即可在目标机器上面进行调用。

. 典型编译方式二

通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=riscv  \
  -DONLY_RUNTIME=ON \
  -DBUILD_PYSAIL=OFF \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_RISCV/ToolChain_
↪riscv64_linux.cmake \
```

(下页继续)

(续上页)

```
  -DLIBSOPHON_BASIC_PATH=libsophon_0.5.0_riscv64/opt/sophon/
↪libsophon-0.5.0 ..
make sail
```

4. 安装 SAIL 动态库及头文件, 程序将自动在源码目录下创建‘build_riscv’, 编译结果将安装在‘build_riscv’下面

```
make install
```

5. 将‘build_riscv’文件夹下的‘sophon-sail’拷贝至目标 RISCV 主机的‘/opt/sophon’目录下, 即可在目标机器上面进行调用。

WINDOWS MODE

. 安装 libsophon,sophon-ffmpeg,sophon-opencv 的 SAIL

windows 下 libsophon,sophon-ffmpeg,sophon-opencv 的安装方式可参考算能官方文档

下载并安装 Visual Studio 2019 作为 windows 下的开发工具

. 典型编译方式一

编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹
3. 执行 cmake 命令，生成项目文件.sln

```
cmake -DBUILD_TYPE=windows -DTARGET_TYPE=release -DRUNTIME_
↪LIB=MT -DBUILD_PYSAIL=OFF -DLIBSOPHON_DIR=C:/sophon_sdk_
↪win_release_MT/libsophon_0.4.9/data -DOPENCV_DIR=C:/sophon_sdk_
↪win_release_MT/sophon-opencv_0.6.0/lib/cmake/opencv4 -DFFMPEG_
↪DIR=C:/sophon_sdk_win_release_MT/sophon-ffmpeg_0.6.0/lib/cmake ..
```

4. 在 vs2019 下打开.sln 项目文件，修改编译模式为 release，点击生成项目。
5. 项目编译成功后会在 build/lib/Release 文件夹下生成 sail.lib,sail.exp,sail.dll 等文件。
6. 安装 sail 库，在 build 路径下运行指令, 生成 build_windows 文件夹

```
cmake --install .
```

7. 在 CMakeLists.txt 文件中引入 sail 库即可，如:

```
set(SAIL_DIR your_path/build_windows/sophon-sail/lib/cmake)
find_package(SAIL REQUIRED)
include_directories(${SAIL_INCLUDE_DIRS})
link_directories(${SAIL_LIB_DIRS})
    ...
target_link_libraries(your_project sail.lib)
```


. 典型编译方式二

编译不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL,

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹
3. 执行 cmake 命令，生成项目文件.sln

```
cmake -DBUILD_TYPE=windows -DTARGET_TYPE=release -DRUNTIME_
↪LIB=MT -DBUILD_PYSAIL=OFF -DONLY_RUNTIME=ON -
↪DLIBSOPHON_DIR=C:/sophon_sdk_win_release_MT/libsophon_0.4.9/data␣
↪..
```

4. 在 vs2019 下打开.sln 项目文件，修改编译模式为 release，点击生成项目。
5. 项目编译成功后会在 build/lib/Release 文件夹下生成 sail.lib,sail.exp,sail.dll 等文件。
6. 安装 sail 库，在 build 路径下运行指令, 生成 build_windows 文件夹

```
cmake --install .
```

7. 在 CMakeLists.txt 文件中引入 sail 库即可，如:

```
set(SAIL_DIR your_path/build_windows/sophon-sail/lib/cmake)
find_package(SAIL REQUIRED)
include_directories(${SAIL_INCLUDE_DIRS})
link_directories(${SAIL_LIB_DIRS})
    ...
target_link_libraries(your_project sail.lib)
```

3.2.3 编译可被 Python3 接口调用的 Wheel 文件

注意: BM1688 和 CV186AH 处理器仅支持 SOC MODE 章节,BM1684 和 BM1684X 处理器无此限制。对于 BM1688 或者 CV186AH 处理器, 下列命令中的 sophon-mw 应改为 sophon-media。

PCIE MODE

. 安装 libsophon,sophon-ffmpeg,sophon-opencv 的 SAIL

libsophon,sophon-ffmpeg,sophon-opencv 的安装方式可参考算能官方文档

. 典型编译方式一

使用默认安装路径, 编译包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为‘python/pcie/dist’, 文件名为‘sophon-3.8.0-py3-none-any.whl’

```
cd ../python/pcie
chmod +x sophon_pcie_whl.sh
./sophon_pcie_whl.sh
```

5. 安装 python wheel

```
pip3 install ./dist/sophon-3.8.0-py3-none-any.whl --force-reinstall
```

. 典型编译方式二

编译不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL,

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DONLY_RUNTIME=ON ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为‘python/pcie/dist’, 文件名为‘sophon-3.8.0-py3-none-any.whl’

```
cd ../python/pcie
chmod +x sophon_pcie_whl.sh
./sophon_pcie_whl.sh
```

5. 安装 python wheel

```
pip3 install ./dist/sophon-3.8.0-py3-none-any.whl --force-reinstall
```

. 典型编译方式三

如果生产环境与开发环境上的 python3 版本不一致, 可以通过升级 python3 版本使其保持一致, 也可以通过 python3 的官方网站获取获取相应的 python3 包, 或者根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。也就是使用非系统默认的 python3, 编译包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, 并打包到‘build_pcie’目录下, 本示例使用的 python3 路径为‘python_3.8.2/bin/python3’,python3 的动态库目录‘python_3.8.2/lib’。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 -DCUSTOM_
↪PY_LIBDIR=python_3.8.2/lib ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为‘python/pcie/dist’, 文件名为‘sophon-3.8.0-py3-none-any.whl’

```
cd ../python/pcie
chmod +x sophon_pcie_whl.sh
./sophon_pcie_whl.sh
```

5. 安装 python wheel

将‘sophon-3.8.0-py3-none-any.whl’拷贝到目标机器上, 然后执行如下安装命令

```
pip3 install ./dist/sophon-3.8.0-py3-none-any.whl --force-reinstall
```

SOC MODE

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.4.1, sophon-ffmpeg 的版本为 0.4.1,sophon-opencv 的版本为 0.4.1。

1. 从算能官网中获取‘libsophon_soc_0.4.1_aarch64.tar.gz’, 并解压

```
tar -xvf libsophon_soc_0.4.1_aarch64.tar.gz
```

解压后 libsophon 的目录为‘libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1’

2. 从算能官网中获取‘sophon-mw-soc_0.4.1_aarch64.tar.gz’, 并解压

```
tar -xvf sophon-mw-soc_0.4.1_aarch64.tar.gz
```

解压后 sophon-ffmpeg 的目录为‘sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1’。

解压后 sophon-opencv 的目录为‘sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1’。

. 安装 g++-aarch64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

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

. 典型编译方式一

使用指定版本的 python3(和目标 SOC 上的 python3 保持一致), 通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。本示例使用的 python3 路径为 'python_3.8.2/bin/python3' ,python3 的动态库目录 'python_3.8.2/lib'。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=soc  \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
   -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
   -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
   -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 \
   -DFFMPEG_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
↪sophon-ffmpeg_0.4.1 \
   -DOPENCV_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
↪sophon-opencv_0.4.1 ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为 'python/soc/dist' , 文件名为 'sophon_arm-3.8.0-py3-none-any.whl'

```
cd ../python/soc
chmod +x sophon_soc_whl.sh
./sophon_soc_whl.sh
```

5. 安装 python wheel

将 'sophon_arm-3.8.0-py3-none-any.whl' 拷贝到目标 SOC 上, 然后执行如下安装命令

```
pip3 install sophon_arm-3.8.0-py3-none-any.whl --force-reinstall
```

. 典型编译方式二

使用指定版本的 python3(和目标 SOC 上的 python3 保持一致), 通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。本示例使用的 python3 路径为 'python_3.8.2/bin/python3' ,python3 的动态库目录 'python_3.8.2/lib'。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=soc \
  -DONLY_RUNTIME=ON \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
  -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
  -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
  -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为 'python/soc/dist', 文件名为 'sophon_arm-3.8.0-py3-none-any.whl'

```
cd ../python/soc
chmod +x sophon_soc_whl.sh
./sophon_soc_whl.sh
```

5. 安装 python wheel

将 'sophon_arm-3.8.0-py3-none-any.whl' 拷贝到目标 SOC 上, 然后执行如下安装命令

```
pip3 install sophon_arm-3.8.0-py3-none-any.whl --force-reinstall
```

ARM PCIE MODE

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.4.1, sophon-ffmpeg 的版本为 0.4.1,sophon-opencv 的版本为 0.4.1。

1. 从算能官网中获取 'libsophon_0.4.1_aarch64.tar.gz', 并解压

```
tar -xvf libsophon_0.4.1_aarch64.tar.gz
```

解压后 libsophon 的目录为 'libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1'

2. 从算能官网中获取 'sophon-mw_0.4.1_aarch64.tar.gz', 并解压

```
tar -xvf sophon-mw_0.4.1_aarch64.tar.gz
```

解压后 sophon-ffmpeg 的目录为 'sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1'。

解压后 sophon-opencv 的目录为 'sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1'。

. 安装 g++-aarch64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

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

. 典型编译方式一

使用指定版本的 python3(和目标 ARM 主机上的 python3 保持一致), 通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。本示例使用的 python3 路径为 'python_3.8.2/bin/python3' ,python3 的动态库目录 'python_3.8.2/lib'。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=arm_pcie  \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
↪ToolChain_aarch64_linux.cmake \
  -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
  -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
  -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 \
  -DFFMPEG_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
↪ffmpeg_0.4.1 \
  -DOPENCV_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
↪opencv_0.4.1 ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为 'python/arm_pcie/dist' , 文件名为 'sophon_arm_pcie-3.8.0-py3-none-any.whl'

```
cd ../python/arm_pcie
chmod +x sophon_arm_pcie_whl.sh
./sophon_arm_pcie_whl.sh
```

5. 安装 python wheel

将 'sophon_arm_pcie-3.8.0-py3-none-any.whl' 拷贝到目标 ARM 主机上, 然后执行如下安装命令

```
pip3 install sophon_arm_pcie-3.8.0-py3-none-any.whl --force-reinstall
```

. 典型编译方式二

使用指定版本的 python3(和目标 ARM 主机上的 python3 保持一致), 通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编

译好的 python3。本示例使用的 python3 路径为‘python_3.8.2/bin/python3’,python3 的动态库目录‘python_3.8.2/lib’。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=arm_pcie  \
   -DONLY_RUNTIME=ON \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
↪ToolChain_aarch64_linux.cmake \
   -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
   -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
   -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为‘python/arm_pcie/dist’, 文件名为‘sophon_arm_pcie-3.8.0-py3-none-any.whl’

```
cd ../python/arm_pcie
chmod +x sophon_arm_pcie_whl.sh
./sophon_arm_pcie_whl.sh
```

5. 安装 python wheel

将‘sophon_arm_pcie-3.8.0-py3-none-any.whl’拷贝到目标 ARM 主机上, 然后执行如下安装命令

```
pip3 install sophon_arm_pcie-3.8.0-py3-none-any.whl --force-reinstall
```

LOONGARCH64 MODE

. 安装 loongarch64-linux-gnu 工具链

从 LoongArch64 官网获取其 [交叉编译的工具链](http://ftp.loongnix.cn/toolchain/gcc/release/loongarch/gcc8/loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1.tar.xz), 解压到本地, 解压后的目录结构如下:

```
└── loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1
    ├── bin
    ├── lib
    ├── lib64
    ├── libexec
    ├── loongarch64-linux-gnu
    ├── share
```

(下页继续)

(续上页)

```
├─── sysroot
└─── versions
```

. 获取交叉编译需要使用的 libsophon

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.4.7。

. 典型编译方式一

使用指定版本的 python3(和目标龙芯主机上的 python3 保持一致), 通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。本示例使用的 python3 路径为 'python_3.7.3/bin/python3' ,python3 的动态库目录 'python_3.7.3/lib'。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=loongarch  \
   -DONLY_RUNTIME=ON \
   -DTOOLCHAIN_BASIC_PATH=toolchains/loongson-gnu-toolchain-8.3-x86_
↪64-loongarch64-linux-gnu-rc1.1 \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_LoongArch64/
↪ToolChain_loongarch64_linux.cmake \
   -DPYTHON_EXECUTABLE=python_3.7.3/bin/python3 \
   -DCUSTOM_PY_LIBDIR=python_3.7.3/lib \
   -DLIBSOPHON_BASIC_PATH=libsophon_0.4.7_loongarch64/opt/sophon/
↪libsophon-0.4.7 \
   ..
make pysail
```

cmake 选项中的路径需要您根据环境的配置进行调整

- DLIBSOPHON_BASIC_PATH: SOPHONSDK 中 libsophon 下对应 libsophon_<x.y.z>_loongarch64.tar.gz 解压后的目录。

4. 打包生成 python wheel, 生成的 wheel 包的路径为 'python/loongarch64/dist' , 文件名为 'sophon_loongarch64-3.8.0-py3-none-any.whl'

```
cd ../python/loongarch64
chmod +x sophon_loongarch64_whl.sh
./sophon_loongarch64_whl.sh
```

注：此处易出现 setuptools 版本过高的问题，原则上 python3.8 最高兼容 setuptools 版本 < 66.0.0

5. 安装 python wheel

将‘sophon_loongarch64-3.8.0-py3-none-any.whl’拷贝到目标主机上, 然后执行如下安装命令

```
pip3 install sophon_loongarch64-3.8.0-py3-none-any.whl --force-reinstall
```

RISCV MODE

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

此章节所有的编译操作都是在 x86 主机上, 使用交叉编译的方式进行编译。下面示例中选择 libsophon 的版本为 0.5.0, sophon-ffmpeg 的版本为 0.6.0,sophon-opencv 的版本为 0.6.0。

1. 从算能官网中获取‘libsophon_0.5.0_riscv64.tar.gz’, 并解压

```
tar -xvf libsophon_0.5.0_riscv64.tar.gz
```

解压后 libsophon 的目录为‘libsophon_0.5.0_riscv64/opt/sophon/libsophon-0.5.0’

2. 从算能官网中获取‘sophon-mw_0.6.0_riscv_64.tar.gz’, 并解压

```
tar -xvf sophon-mw_0.6.0_riscv_64.tar.gz
```

解压后 sophon-ffmpeg 的目录为‘sophon-mw_0.6.0_riscv_64/opt/sophon/sophon-ffmpeg_0.6.0’。

解压后 sophon-opencv 的目录为‘sophon-mw_0.6.0_riscv_64/opt/sophon/sophon-opencv_0.6.0’。

. 安装 g++-riscv64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

```
sudo apt-get install gcc-riscv64-linux-gnu g++-riscv64-linux-gnu
```

. 典型编译方式一

使用指定版本的 python3(和目标 RISCV 服务器上的 python3 保持一致), 通过交叉编译的方式, 编译出包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。本示例使用的 python3 路径为‘python_3.11.0/bin/python3’,python3 的动态库目录‘python_3.11.0/lib’。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=riscv  \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_RISCV/ToolChain_
↪riscv64_linux.cmake \
   -DPYTHON_EXECUTABLE=python_3.11.0/bin/python3 \
   -DCUSTOM_PY_LIBDIR=python_3.11.0/lib \
   -DLIBSOPHON_BASIC_PATH=libsophon_0.5.0_riscv64/opt/sophon/
↪libsophon-0.5.0 \
   -DFFMPEG_BASIC_PATH=sophon-mw_0.6.0_riscv_64/opt/sophon/
↪sophon-ffmpeg_0.6.0 \
   -DOPENCV_BASIC_PATH=sophon-mw_0.6.0_riscv_64/opt/sophon/
↪sophon-opencv_0.6.0 ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为‘python/riscv/dist’, 文件名为‘sophon_riscv64-3.8.0-py3-none-any.whl’

```
cd ../python/riscv
chmod +x sophon_riscv_whl.sh
./sophon_riscv_whl.sh
```

5. 安装 python wheel

将‘sophon_riscv64-3.8.0-py3-none-any.whl’拷贝到目标 RISCV 服务器上, 然后执行如下安装命令

```
pip3 install sophon_riscv64-3.8.0-py3-none-any.whl --force-reinstall
```

. 典型编译方式二

使用指定版本的 python3(和目标 RISCV 服务器上的 python3 保持一致), 通过交叉编译的方式, 编译出不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL, python3 的安装方式可通过 python 官方网站获取, 也可以根据 [获取在 X86 主机上进行交叉编译的 Python3] 获取已经编译好的 python3。本示例使用的 python3 路径为‘python_3.11.0/bin/python3’,python3 的动态库目录‘python_3.11.0/lib’。

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹

```
mkdir build && cd build
```

3. 执行编译命令

```
cmake -DBUILD_TYPE=riscv  \
   -DONLY_RUNTIME=ON \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_RISCV/ToolChain_
↪riscv64_linux.cmake \
   -DPYTHON_EXECUTABLE=python_3.11.0/bin/python3 \
   -DCUSTOM_PY_LIBDIR=python_3.11.0/lib \
```

(下页继续)

(续上页)

```
  -DLIBSOPHON_BASIC_PATH=libsophon_0.5.0_riscv64/opt/sophon/
↪libsophon-0.5.0 ..
make pysail
```

4. 打包生成 python wheel, 生成的 wheel 包的路径为‘python/riscv/dist’，文件名为‘sophon_riscv64-3.8.0-py3-none-any.whl’

```
cd ../python/riscv
chmod +x sophon_riscv_whl.sh
./sophon_riscv_whl.sh
```

5. 安装 python wheel

将‘sophon_riscv64-3.8.0-py3-none-any.whl’拷贝到目标 RISCV 服务器上, 然后执行如下安装命令

```
pip3 install sophon_riscv64-3.8.0-py3-none-any.whl --force-reinstall
```

WINDOWS MODE

. 安装 libsophon,sophon-ffmpeg,sophon-opencv 的 SAIL

windows 下 libsophon,sophon-ffmpeg,sophon-opencv 的安装方式可参考算能官方文档

. 典型编译方式一

编译包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹
3. 执行 cmake 指令，生成项目文件（.sln）

```
cmake -DBUILD_PYSAIL=ON -DBUILD_TYPE=windows -DTARGET_
↪TYPE=release -DRUNTIME_LIB=MT -DLIBSOPHON_DIR=C:/sophon_
↪sdk_win_release_MT/libsophon_0.4.9/data -DOPENCV_DIR=C:/sophon_
↪sdk_win_release_MT/sophon-opencv_0.6.0/lib/cmake/opencv4 -DFFMPEG_
↪DIR=C:/sophon_sdk_win_release_MT/sophon-ffmpeg_0.6.0/lib/cmake -
↪DPYTHON_EXECUTABLE=C:\Users\SOPHGO\AppData\Local\Programs\
↪Python\Python38\python.exe ..
```

4. 在 vs2019 下打开.sln 项目文件，修改编译模式为 release，点击生成项目。
5. 在 sophon-sail/python/windows 路径下运行指令 python setup.py bdist_wheel 打包生成 dist 文件夹下的 wheel 包，文件名为‘sophon-3.8.0-py3-none-any.whl’
6. 安装 python wheel

```
pip3 install ./dist/sophon-3.8.0-py3-none-any.whl --force-reinstall
```

python3.8 及以上版本需要显示添加 DLL 动态库路径才能成功导入 sail 包，例：

```
import os
os.add_dll_directory(r'C:\sophon_sdk_win_release_MT\libsophon_0.4.9\bin')
os.add_dll_directory(r'C:\sophon_sdk_win_release_MT\sophon-ffmpeg_0.6.0\lib')
os.add_dll_directory(r'C:\sophon_sdk_win_release_MT\sophon-opencv_0.6.0\lib')
import sophon.sail as sail
```

. 典型编译方式二

编译不包含 bmcv,sophon-ffmpeg,sophon-opencv 的 SAIL,

通过此方式编译出来的 SAIL 无法使用其 Decoder、Bmcv 等多媒体相关接口。

1. 下载 SOPHON-SAIL 源码, 解压后进入其源码目录
2. 创建编译文件夹 build, 并进入 build 文件夹
3. 执行 cmake 指令，生成项目文件（.sln）

```
cmake -DONLY_RUNTIME=ON -DBUILD_PYSAIL=ON -DBUILD_
↪TYPE=windows -DTARGET_TYPE=release -DRUNTIME_LIB=MT -
↪DLIBSOPHON_DIR=C:/sophon_sdk_win_release_MT/libsophon_0.4.9/data␣
↪-DPYTHON_EXECUTABLE=C:\\Users\\SOPHGO\\AppData\\Local\\
↪Programs\\Python\\Python38\\python.exe ..
```

4. 在 vs2019 下打开.sln 项目文件，修改编译模式为 release，点击生成项目。
5. 在 sophon-sail/python/windows 路径下运行指令 python setup.py bdist_wheel 打包生成 dist 文件夹下的 wheel 包，文件名为‘sophon-3.8.0-py3-none-any.whl’
6. 安装 python wheel

```
pip3 install ./dist/sophon-3.8.0-py3-none-any.whl --force-reinstall
```

python3.8 及以上版本需要显示添加 DLL 动态库路径才能成功导入 sail 包，例：

```
import os
os.add_dll_directory(r'C:\sophon_sdk_win_release_MT\libsophon_0.4.9\bin')
os.add_dll_directory(r'C:\sophon_sdk_win_release_MT\sophon-ffmpeg_0.6.0\lib')
os.add_dll_directory(r'C:\sophon_sdk_win_release_MT\sophon-opencv_0.6.0\lib')
import sophon.sail as sail
```

3.2.4 编译用户手册

. 安装软件包

```
# 更新apt
sudo apt update
# 安装latex
sudo apt install texlive-xetex texlive-latex-recommended
# 安装Sphinx
pip3 install sphinx sphinx-autobuild sphinx_rtd_theme rst2pdf
# 安装结巴中文分词库，以支持中文搜索
pip3 install jieba3k
```

. 安装字体

[Fandol](https://ctan.org/pkg/fandol) - Four basic fonts for Chinese typesetting

```
# 下载Fandol字体
wget http://mirrors.ctan.org/fonts/fandol.zip
# 解压缩字体包
unzip fandol.zip
# 拷贝安装字体包
sudo cp -r fandol /usr/share/fonts/
cp -r fandol ~/.fonts
```

. 执行编译

下载 SOPHON-SAIL 源码, 解压后进入其源码的 docs 目录

```
cd docs
make pdf
```

编译好的用户手册路径为‘docs/build/sophon-sail_zh.pdf’

如果编译仍然报错, 可以安装以下‘sudo apt-get install texlive-lang-chinese’, 然后重新执行上述命令。

3.3 使用 SAIL 的 Python 接口进行开发

注意: BM1688 和 CV186AH 处理器仅支持 SOC MODE 章节,BM1684 和 BM1684X 处理器无此限制。对于 BM1688 或者 CV186AH 处理器, 下列命令中的 sophon-mw 应改为 sophon-media。

3.3.1 PCIE MODE

在使用 PCIE MODE 编译好 SAIL, 执行安装 python wheel 之后, 即可以使用 python 中调用 SAIL, 其接口文档可参考 API 章节。

3.3.2 SOC MODE

. 使用自己编译的 Python wheel 包

在使用 SOC MODE 通过交叉编译的方式编译好 SAIL 之后, 将 python wheel 拷贝到 SOC 上面进行安装, 即可以使用 python 中调用 SAIL, 其接口文档可参考 API 章节。

. 使用预编译的 Python wheel 包

1. 查看 SOC 上的 libsophon 版本和 sophon-mw(sophon-ffmpeg,sophon-opencv) 的版本

   ```
   ls /opt/sophon/
   ```

2. 查看 SOC 上的 Python3 版本

```
python3 --version
```

3. 从预编译的 Python wheel 包中找到对应版本的 wheel 包, 将对应的 wheel 包拷贝到 SOC 上面进行安装, 即可以使用 python 中调用 SAIL, 其接口文档可参考 API 章节。

3.3.3 ARM PCIE MODE

在使用 ARM PCIE MODE 通过交叉编译的方式编译好 SAIL 之后, 将 python wheel 拷贝到 ARM 主机上面进行安装, 即可以在 python 中调用 SAIL, 其接口文档可参考 API 章节。

1. 查看 ARM 主机上的 libsophon 版本和 sophon-mw(sophon-ffmpeg,sophon-opencv) 的版本

```
cat /opt/sophon/
```

2. 查看 ARM 主机上的 Python3 版本

```
python3 --version
```

3. 从预编译的 Python wheel 包中找到对应版本的 wheel 包, 将对应的 wheel 包拷贝到 ARM 主机上面进行安装, 即可以使用 python 中调用 SAIL, 其接口文档可参考 API 章节。

3.4 使用 SAIL 的 C++ 接口进行开发

注意: BM1688 和 CV186AH 处理器仅支持 SOC MODE 章节,BM1684 和 BM1684X 处理器无此限制。对于 BM1688 或者 CV186AH 处理器, 下列命令中的 sophon-mw 应改为 sophon-media。

3.4.1 PCIE MODE

在使用 PCIE MODE 编译好 SAIL, 并且通过执行 'sudo make install' 或者通过拷贝的方式安装好 SAIL 的 c++ 库之后, 推荐使用 cmake 来将 SAIL 中的库链接到自己的程序中, 如果需要使用 SAIL 多媒体相关的功能, 也需要将 libsophon,sophon-ffmpeg,sophon-opencv 的头文件目录及动态库目录添加到自己的程序中。可以在您程序的 CMakeLists.txt 中添加如下段落:

```
find_package(libsophon REQUIRED)
include_directories(${LIBSOPHON_INCLUDE_DIRS})
# 添加libsophon的头文件目录

set(SAIL_DIR  /opt/sophon/sophon-sail/lib/cmake)
find_package(SAIL REQUIRED)
include_directories(${SAIL_INCLUDE_DIRS})
link_directories(${SAIL_LIB_DIRS})
# 添加SAIL的头文件及动态库目录
```

(下页继续)

(续上页)

```
find_package(libsophon REQUIRED)
include_directories(${LIBSOPHON_INCLUDE_DIRS})
# 添加libsophon的头文件目录

set(OpenCV_DIR  /opt/sophon/sophon-opencv-latest/lib/cmake/opencv4)
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})
# 添加sophon-opencv的头文件目录

set(FFMPEG_DIR  /opt/sophon/sophon-ffmpeg-latest/lib/cmake)
find_package(FFMPEG REQUIRED)
include_directories(${FFMPEG_INCLUDE_DIRS})
link_directories(${FFMPEG_LIB_DIRS})
# 添加sophon-ffmpeg的头文件及动态库目录

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})
target_link_libraries(${YOUR_TARGET_NAME} sail)
```

在您的代码中即可以调用 sail 中的函数:

```
#define USE_FFMPEG  1
#define USE_OPENCV  1
#define USE_BMCV    1

#include <stdio.h>
#include <sail/cvwrapper.h>
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int device_id = 0;
    std::string video_path = "test.avi";
    sail::Decoder decoder(video_path,true,device_id);
    if(!decoder.is_opened()){
        printf("Video[%s] read failed!\n",video_path.c_str());
        exit(1) ;
    }

    sail::Handle handle(device_id);
    sail::Bmcv bmcv(handle);

    while(true){
        sail::BMImage ost_image = decoder.read(handle);
        bmcv.imwrite("test.jpg", ost_image);
        break;
    }
```

(下页继续)

(续上页)

```
    return 0;
}
```

3.4.2 SOC MODE

.SOC 板卡上编译程序

在 SOC 板卡上安装好 libsophon,sophon-ffmpeg,sophon-opencv, 及 SAIL 之后, 您可以参考 PCIE MODE 的开发方法使用 cmake 将 SAIL 中的库链接到自己的程序中, 如果需要使用 SAIL 多媒体相关的功能, 也需要将 libsophon,sophon-ffmpeg,sophon-opencv 的头文件目录及动态库目录添加到自己的程序中。

.x86 交叉编译程序

如果您希望使用 SAIL 搭建交叉编译环境, 您需要用到 libsophon,sophon-ffmpeg,sophon-opencv 以及 gcc-aarch64-linux-gnu 工具链。

. 创建‘soc-sdk’文件夹

创建‘soc-sdk’文件夹, 后续交叉编译需要用到的头文件及动态库都将存放在此目录中。

```
mkdir soc-sdk
```

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

下面示例中选择 libsophon 的版本为 0.4.1, sophon-ffmpeg 的版本为 0.4.1,sophon-opencv 的版本为 0.4.1。

1. 从算能官网中获取‘libsophon_soc_0.4.1_aarch64.tar.gz’, 并解压拷贝至‘soc-sdk’文件夹

```
tar -xvf libsophon_soc_0.4.1_aarch64.tar.gz
cp -r libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/include soc-sdk
cp -r libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/lib soc-sdk
```

解压后 libsophon 的目录为‘libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1’

2. 从算能官网中获取‘sophon-mw-soc_0.4.1_aarch64.tar.gz’, 并解压拷贝至‘soc-sdk’文件夹

```
tar -xvf sophon-mw-soc_0.4.1_aarch64.tar.gz
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/include␣
↪soc-sdk
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/lib soc-sdk
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/include/
↪opencv4/opencv2 soc-sdk/include
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/lib soc-sdk
```

. 将交叉编译好的 SAIL, 也即是‘build_soc’拷贝至‘soc-sdk’文件夹

```
cp build_soc/sophon-sail/include soc-sdk
cp build_soc/sophon-sail/lib soc-sdk
```

. 安装 g++-aarch64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

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

上述步骤配置好之后, 可以通过配置 cmake 来完成交叉编译, 在您程序的 CMakeLists.txt 中添加如下段落:

CMakeLists.txt 中需要使用‘soc-sdk’的绝对路径为‘/opt/sophon/soc-sdk’, 实际应用中需要根据自己实际的位置来进行配置。

```
set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_ASM_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)

include_directories("/opt/sophon/soc-sdk/include")
include_directories("/opt/sophon/soc-sdk/include/sail")
# 添加交叉编译需要使用的头文件目录

link_directories("/opt/sophon/soc-sdk/lib")
# 添加交叉编译需要使用的动态库目录

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})
target_link_libraries(${YOUR_TARGET_NAME} sail)
# sail为需要链接的库
```

3.4.3 ARM PCIE MODE

.ARM 主机上编译程序

在 ARM 主机上安装好 libsophon,sophon-ffmpeg,sophon-opencv, 及 SAIL 之后, 您可以参考 PCIE MODE 的开发方法使用 cmake 将 SAIL 中的库链接到自己的程序中, 如果需要使用 SAIL 多媒体相关的功能, 也需要将 libsophon,sophon-ffmpeg,sophon-opencv 的头文件目录及动态库目录添加到自己的程序中。

.x86 交叉编译程序

如果您希望使用 SAIL 搭建交叉编译环境, 您需要用到 libsophon,sophon-ffmpeg,sophon-opencv 以及 gcc-aarch64-linux-gnu 工具链。

. 创建‘arm_pcie-sdk’文件夹

创建‘arm_pcie-sdk’文件夹, 后续交叉编译需要用到的头文件及动态库都将存放在此目录中。

```
mkdir arm_pcie-sdk
```

. 获取交叉编译需要使用的 libsophon,sophon-ffmpeg,sophon-opencv

下面示例中选择 libsophon 的版本为 0.4.1, sophon-ffmpeg 的版本为 0.4.1,sophon-opencv 的版本为 0.4.1。

1. 从算能官网中获取‘libsophon_0.4.1_aarch64.tar.gz’, 并解压拷贝至‘arm_pcie-sdk’文件夹

```
tar -xvf libsophon_0.4.1_aarch64.tar.gz
cp -r libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/include arm_pcie-sdk
cp -r libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/lib arm_pcie-sdk
```

解压后 libsophon 的目录为‘libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1’

2. 从算能官网中获取‘sophon-mw_0.4.1_aarch64.tar.gz’, 并解压拷贝至‘arm_pcie-sdk’文件夹

```
tar -xvf sophon-mw_0.4.1_aarch64.tar.gz
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/include arm_
↪pcie-sdk
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/lib arm_pcie-
↪sdk
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/include/
↪opencv4/opencv2 arm_pcie-sdk/include
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/lib arm_pcie-
↪sdk
```

. 将交叉编译好的 SAIL, 也即是‘build_arm_pcie’拷贝至‘arm_pcie-sdk’文件夹

```
cp build_arm_pcie/sophon-sail/include arm_pcie-sdk
cp build_arm_pcie/sophon-sail/lib arm_pcie-sdk
```

. 安装 g++-aarch64-linux-gnu 工具链

如果已经安装, 可忽略此步骤

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

上述步骤配置好之后, 可以通过配置 cmake 来完成交叉编译, 在您程序的 CMakeLists.txt 中添加如下段落:

CMakeLists.txt 中需要使用‘arm_pcie-sdk’的绝对路径为‘/opt/sophon/arm_pcie-sdk’, 实际应用中需要根据自己实际的位置来进行配置。

```
set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_ASM_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)

include_directories("/opt/sophon/arm_pcie-sdk/include")
include_directories("/opt/sophon/arm_pcie-sdk/include/sail")
# 添加交叉编译需要使用的头文件目录
```

(下页继续)

(续上页)

```
link_directories("/opt/sophon/arm_pcie-sdk/lib")
# 添加交叉编译需要使用的动态库目录

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})
target_link_libraries(${YOUR_TARGET_NAME} sail)
# sail为需要链接的库
```

CHAPTER 4

SAIL C++ API 参考

4.1 Basic function

主要用于获取或配置设备信息与属性。

4.1.1 get_available_tpu_num

获取当前设备中可用智能视觉深度学习处理器的数量。

接口形式:

```
int get_available_tpu_num();
```

返回值说明:

返回当前设备中可用智能视觉深度学习处理器的数量。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int tpu_len = sail::get_available_tpu_num();
    std::cout << "available tpu: " << tpu_len << std::endl;
    return 0;
}
```

4.1.2 set_print_flag

设置是否打印程序的计算耗时信息。

接口形式:

```
int set_print_flag(bool print_flag);
```

参数说明:

- print_flag: bool

print_flag 为 True 时，打印程序的计算主要的耗时信息，否则不打印。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int ret = sail::set_print_flag(true);
    if (ret == 0){
        std::cout << "set print time success" << std::endl;
    }
    return 0;
}
```

4.1.3 set_dump_io_flag

设置是否存储输入数据和输出数据。

接口形式:

```
int set_dump_io_flag(bool dump_io_flag);
```

参数说明:

- dump_io_flag: bool

dump_io_flag 为 True 时，存储输入数据和输出数据，否则不存储。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    ret = sail::set_dump_io_flag(true);
    if (ret == 0){
        std::cout << "set save data success" << std::endl;
    }
    return 0;
}
```

4.1.4 set_loglevel

设置运行过程中的日志级别为指定级别。较低的日志等级通常用于生产环境，以减少性能开销和日志数据量，而较高的日志等级则适用于开发和调试，以便能够记录更详细的信息。

接口形式:

```
int set_loglevel(LogLevel loglevel);
```

参数说明:

- loglevel: LogLevel

期望的日志级别，为 sail::LogLevel 枚举值。可选的级别包括 TRACE、DEBUG、INFO、WARN、ERROR、CRITICAL、OFF，默认级别为 INFO。

返回值说明:

返回类型：int

0：日志级别设置成功。-1：传入了未知的日志级别，设置失败。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int ret = sail::set_loglevel(sail::LogLevel::TRACE);
    if (ret == 0){
        std::cout << "Set log level successfully" << std::endl;
    }
    else{
        std::cout << "Unknown log level, set failed." << std::endl;
    }
    return 0;
}
```

4.1.5 set_decoder_env

设置 Decoder 的环境变量，必须在 Decoder 构造前设置，否则使用默认值。

接口形式:

```
int set_decoder_env(std::string env_name, std::string env_value);
```

参数说明:

- env_name: string

选择设置 Decoder 的属性名称，可选的属性名称有:

- ‘refcounted_frames’ 设置为 1 时，解码出来的图像需要程序手动释放，为 0 时由 Decoder 自动释放。

- ‘extra_frame_buffer_num’ 设置 Decoder 的最大缓存帧数
- ‘rtsp_transport’ 设置 RTSP 采用的传输协议
- ‘stimeout’ 设置阻塞超时时间
- ‘rtsp_flags’ 设置 RTSP 是否自定义 IO
- ‘buffer_size’ 设置缓存大小
- ‘max_delay’ 设置最大时延
- ‘probesize’ 解析文件时读取的最大字节数
- ‘analyzeduration’ 解析文件时读取的最大时长

- env_value: string

该属性的配置值

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    sail::set_decoder_env("extra_frame_buffer_num", "1");
    int dev_id = 0;
    sail::Handle handle(dev_id);
    int width = 640;
    int height = 640;
    sail::Format format = sail::Format::FORMAT_BGR_PLANAR;
    sail::ImgDtype dtype = sail::ImgDtype::DATA_TYPE_EXT_1N_BYTE;
    sail::BMImage bmimg1(handle, width, height, format, dtype);
    std::string image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    sail::Decoder decoder(image_name, false, dev_id);
    decoder.read(handle, bmimg1);

    return 0;
}
```

4.1.6 base64_enc

对数据进行 base64 编码，生成的对应的 base64 编码后的字符串。

接口形式:

```
int base64_enc(Handle& handle, const void *data, uint32_t dlen, std::string& encoded);
```

参数说明:

- handle: Handle

设备的 handle 句柄，使用 Handle(dev_id) 创建

- data: void*

待编码的数据指针

- dlen: uint32_t

待编码的数据字节长度

- encoded: string

base64 编码生成的字符串

返回值说明

base64 编码成功返回 0，否则返回-1

示例代码:

```
#include <sail/base64.h>

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);

    std::string data = "hello,world!";

    // base64 encode
    std::string base64_encoded;
    uint32_t dlen = data.length();
    ret = sail::base64_enc(handle, data.c_str(), dlen, base64_encoded);
    if (ret == 0){
        std::cout << dlen << std::endl;
        std::cout << "base64 encode success!" << "based 64:" << base64_encoded << " lens
↪" << dlen << std::endl;
    }
    return 0;
}
```

4.1.7 base64_dec

对数据进行 base64 编码，生成的对应的 base64 编码后的字符串。示例代码请参考 base64_dec 接口用法。

接口形式:

```
int base64_dec(Handle& handle, const void *data, uint32_t dlen, uint8_t* p_outbuf,␣
↪uint32_t *p_size);
```

参数说明:

- handle: Handle

设备的 handle 句柄，使用 Handle(dev_id) 创建

- data: void*

待解码的数据指针

- dlen: uint32_t

待解码的数据字节长度

- p_outbuf: uint8_t*

解码后的数据 buffer

- p_size: uint32_t

输出数据。解码后的数据指针长度

返回值说明

base64 解码成功返回 0，否则返回-1

示例代码:

```
#include <sail/base64.h>

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);

    std::string data = "hello,world!";

    // base64 encode
    std::string base64_encoded;
    uint32_t dlen = data.length();
    ret = sail::base64_enc(handle, data.c_str(), dlen, base64_encoded);
    if (ret == 0){
        std::cout << dlen << std::endl;
        std::cout << "base64 encode success!" << "based 64:" << base64_encoded << "lens"
↪<< dlen << std::endl;
    }

    // base64_dec
    uint32_t dlen_based = base64_encoded.length();
    uint8_t out_data_buf[100]; // 假设有足够大的空间存放解码后的数据
    uint32_t out_data_size; // 用于存放解码后数据的长度
    ret =sail::base64_dec(handle, base64_encoded.c_str(), dlen_based, out_data_buf, &out_
↪data_size);
    if (ret == 0){
        std::cout << "base64 decode success,data size is:" << out_data_size << std::endl;
        for(uint32_t i = 0; i < out_data_size; i++) {
            std::cout << out_data_buf[i];
        }
        std::cout << std::endl;
    }
    return 0;
}
```

4.1.8 get_tpu_util

获取对应设备的处理器使用率

接口形式:

```
int get_tpu_util(int dev_id);
```

参数说明:

- dev_id: int

需要获取处理器使用率的设备的 ID。

返回值说明:

返回对应设备的处理器的使用率百分比。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int tpu_util;
    tpu_util = sail::get_tpu_util(0); //获取dev0的处理器使用率
    std::cout << "tpu_util " << tpu_util << "%"<< std::endl;
    return 0;
}
```

4.1.9 get_vpu_util

获取对应设备的 VPU 使用率

接口形式:

```
std::vector<int> get_vpu_util(int dev_id);
```

参数说明:

- dev_id: int

需要获取 VPU 使用率的设备的 ID。

返回值说明:

bm1684 为 5 核 vpu，返回值为长度为 5 的 List，bm1684x 为 3 核 vpu，返回值为长度为 3 的 List。List 中的每项数据为对应核心的使用率百分比。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
```

(下页继续)

(续上页)

```
    std::vector<int> vpu_util;
    vpu_util = sail::get_vpu_util(0); //获取dev0的vpu处理器使用率

    for(int i = 0; i < vpu_util.size(); i++) {
        std::cout << "VPU ID: " << i << ", Util Value: " << vpu_util[i] << "%" <<␣
↪std::endl;
    }
    return 0;
}
```

4.1.10 get_vpp_util

获取对应设备的 VPP 使用率

接口形式:

```
std::vector<int> get_vpp_util(int dev_id);
```

参数说明:

- dev_id: int

需要获取 VPP 使用率的设备的 ID。

返回值说明:

bm1684 与 bm1684x 均为 2 核 vpp，返回值为长度为 2 的 List。List 中的每项数据为对应核心的使用率百分比。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    std::vector<int> vpp_util;
    vpp_util = sail::get_vpu_util(0); //获取dev0的vpu处理器使用率

    for(int i = 0; i < vpp_util.size(); i++) {
        std::cout << "VPU ID: " << i << ", Util Value: " << vpp_util[i] << "%" <<␣
↪std::endl;
    }
    return 0;
}
```

4.1.11 get_board_temp

接口形式:

```
int get_board_temp(int dev_id);
```

参数说明:

- dev_id: int

需要获取对应板卡所在设备的 ID。

返回值说明:

返回对应板卡的板级温度，默认单位摄氏度（℃）

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int board_temp;
    board_temp = sail::get_board_temp(0);
    std::cout << "board_temp " << board_temp << "℃"<< std::endl;
    return 0;
}
```

4.1.12 get_chip_temp

接口形式:

```
int get_chip_temp(int dev_id);
```

参数说明:

- dev_id: int

需要获取对应板卡所在设备的 ID。

返回值说明:

返回对应设备的处理器的温度，默认单位摄氏度（℃）。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int chip_temp;
    chip_temp = sail::get_chip_temp(0);
    std::cout << "chip_temp " << chip_temp << "℃"<< std::endl;
    return 0;
}
```

4.1.13 get_dev_stat

接口形式:

```
std::vector<int> get_dev_stat(int dev_id);
```

参数说明:

- dev_id: int

需要获取对应板卡所在设备的 ID。

返回值说明:

返回对应设备的内存信息列表:[mem_total,mem_used,tpu_util]。

示例代码:

```
#include <iostream>
#include "cvwrapper.h"

int main() {
    std::vector<int> dev_stat;
    dev_stat = sail::get_dev_stat(0);

    std::cout << "mem_total: " << dev_stat[0] << " MB" << std::endl;
    std::cout << "mem_used: " << dev_stat[1] << " MB" << std::endl;
    std::cout << "tpu_util: " << dev_stat[2] << " %" << std::endl;
    return 0;
}
```

4.2 Data type

定义 SOPHON 环境中常用的数据类型

接口形式:

```
enum bm_data_type_t {
    BM_FLOAT32,
    BM_INT8,
    BM_UINT8,
    BM_INT32,
    BM_UINT32,
    BM_FLOAT16,
    BM_BFLOAT16,
    BM_INT16,
    BM_UINT16
};
```

参数说明:

- BM_FLOAT32 数据类型为 float32

- BM_INT8 数据类型为 int8
- BM_UINT8 数据类型为 uint8
- BM_INT32 数据类型为 int32
- BM_UINT32 数据类型为 uint32
- BM_FLOAT16 数据类型为 uint32
- BM_BFLOAT16 数据类型为 uint32
- BM_INT16 数据类型为 uint32
- BM_UINT16 数据类型为 uint32

4.3 PaddingAtrr

PaddingAtrr 中存储了数据 padding 的各项属性，可通过配置 PaddingAtrr 进行数据填充

```
class PaddingAtrr {
public:
        PaddingAtrr(){};
        PaddingAtrr(
            unsigned int crop_start_x,
            unsigned int crop_start_y,
            unsigned int crop_width,
            unsigned int crop_height,
            unsigned char padding_value_r,
            unsigned char padding_value_g,
            unsigned char padding_value_b);
    PaddingAtrr(const PaddingAtrr& other);
    ~PaddingAtrr(){};
    void set_stx(unsigned int stx);
    void set_sty(unsigned int sty);
    void set_w(unsigned int w);
    void set_h(unsigned int h);
    void set_r(unsigned int r);
    void set_g(unsigned int g);
    void set_b(unsigned int b);

    unsigned int    dst_crop_stx; // Offset x information relative to the origin of dst␣
↪image
    unsigned int    dst_crop_sty; // Offset y information relative to the origin of dst␣
↪image
    unsigned int    dst_crop_w;   // The width after resize
    unsigned int    dst_crop_h;   // The height after resize
    unsigned char   padding_r;    // Pixel value information of R channel
    unsigned char   padding_g;    // Pixel value information of G channel
    unsigned char   padding_b;    // Pixel value information of B channel
};
```

4.3.1 构造函数 PaddingAtrr()

初始化 PaddingAtrr

接口形式:

```
PaddingAtrr()

PaddingAtrr(
        unsigned int crop_start_x,
        unsigned int crop_start_y,
        unsigned int crop_width,
        unsigned int crop_height,
        unsigned char padding_value_r,
        unsigned char padding_value_g,
        unsigned char padding_value_b);
```

参数说明:

- crop_start_x: int

原图像相对于目标图像在 x 方向上的偏移量

- crop_start_y: int

原图像相对于目标图像在 y 方向上的偏移量

- crop_width: int

在 padding 的同时可对原图像进行 resize，width 为原图像 resize 后的宽，若不进行 resize，则 width 为原图像的宽

- crop_height: int

在 padding 的同时可对原图像进行 resize，height 为原图像 resize 后的高，若不进行 resize，则 height 为原图像的高

- padding_value_r: int

padding 时在 R 通道上填充的像素值

- padding_value_g: int

padding 时在 G 通道上填充的像素值

- padding_value_b: int

padding 时在 B 通道上填充的像素值

4.3.2 set_stx

设置原图像相对于目标图像在 x 方向上的偏移量

接口形式:

```
void set_stx(unsigned int stx);
```

参数说明:

- stx: int

原图像相对于目标图像在 x 方向上的偏移量

4.3.3 set_sty

设置原图像相对于目标图像在 y 方向上的偏移量

接口形式:

```
void set_sty(unsigned int sty);
```

参数说明:

- sty: int

原图像相对于目标图像在 y 方向上的偏移量

4.3.4 set_w

设置原图像 resize 后的 width

接口形式:

```
void set_w(unsigned int w);
```

参数说明:

- width: int

在 padding 的同时可对原图像进行 resize，width 为原图像 resize 后的宽，若不进行 resize，则 width 为原图像的宽

4.3.5 set_h

设置原图像 resize 后的 height

接口形式:

```
void set_h(unsigned int h);
```

参数说明:

- height: int

在 padding 的同时可对原图像进行 resize，height 为原图像 resize 后的高，若不进行 resize，则 height 为原图像的高

4.3.6 set_r

设置 R 通道上的 padding 值

接口形式:

```
void set_r(unsigned int r);
```

参数说明

- r: int

R 通道上的 padding 值

4.3.7 set_g

设置 G 通道上的 padding 值

接口形式:

```
void set_g(unsigned int g);
```

参数说明:

- g: int

G 通道上的 padding 值

4.3.8 set_b

设置 B 通道上的 padding 值

接口形式:

```
void set_b(unsigned int b);
```

参数说明

- b: int

B 通道上的 padding 值

示例代码:

```
#include <stdio.h>
#include <sail/cvwrapper.h>
#include <iostream>
#include <string>

using namespace std;

int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg3 = bmcv.crop_and_resize(BMimg, 0, 0, BMimg.width(), BMimg.
↪height(), 640, 640, paddingatt);
    bmcv.imwrite("{}-{}.jpg".format(BMimg3.width(), BMimg3.height()), BMimg3);
    return 0;
}
```

4.4 Handle

Handle 是设备句柄的包装类，在程序中用于设备的标识。

4.4.1 构造函数 Handle()

初始化 Handle

接口形式:

```
Handle(int tpu_id);
```

参数说明:

- tpu_id: int

创建 Handle 使用的智能视觉深度学习处理器的 id 号

4.4.2 get_device_id

获取 Handle 中智能视觉深度学习处理器的 id

接口形式:

```
int get_device_id();
```

返回值说明:

- tpu_id: int

Handle 中的智能视觉深度学习处理器的 id 号

4.4.3 get_sn

获取 Handle 中标识设备的序列码

接口形式:

```
std::string get_sn();
```

返回值说明:

- serial_number: string

返回 Handle 中设备的序列码

4.4.4 get_target

获取设备的智能视觉深度学习处理器型号

接口形式:

```
std::string get_target();
```

返回值说明:

- Tensor Computing Processor type: str

返回设备智能视觉深度学习处理器的型号

示例代码:

```
#include <stdio.h>
#include <sail/cvwrapper.h>
#include <iostream>

using namespace std;

int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);

    std::cout << "Device ID: " << handle.get_device_id() << std::endl;
    std::cout << "SN: " << handle.get_sn() << std::endl;
    std::cout << "Target: " << handle.get_target() << std::endl;

    return 0;
}
```

4.5 IOMode

IOMode 用于定义输入 Tensor 和输出 Tensor 的内存位置信息（device memory 或 system memory）。

接口形式:

```
enum IOMode {
    SYSI,
    SYSO,
    SYSIO,
    DEVIO
};
```

参数说明:

- SYSI

输入 Tensor 在 system memory，输出 Tensor 在 device memory

- SYSO

输入 Tensor 在 device memory，输出 Tensor 在 system memory

- SYSIO

输入 Tensor 在 system memory，输出 Tensor 在 system memory

- DEVIO

输入 Tensor 在 device memory，输出 Tensor 在 device memory

4.6 LogLevel

LogLevel 用于定义日志级别。最高为 TRACE ， 最低为 OFF ， 默认为 INFO 。

接口形式:

```
enum class LogLevel: int
{
   TRACE,
   DEBUG,
   INFO,
   WARN,
   ERROR,
   CRITICAL,
   OFF
};
```

参数说明:

- TRACE

打印 TRACE 级别和更低级别的日志。

- DEBUG

打印 DEBUG 级别和更低级别的日志。

- INFO

打印 INFO 级别和更低级别的日志。

- WARN

打印 WARN 级别和更低级别的日志。

- ERROR

打印 ERROR 级别和更低级别的日志。

- CRITICAL

打印 CRITICAL 级别和更低级别的日志。

- OFF

关闭各个级别的日志打印。

4.7 bmcv_resize_algorithm

定义图像 resize 中常见的插值策略

接口形式:

```
enum bmcv_resize_algorithm_ {
    BMCV_INTER_NEAREST = 0,
    BMCV_INTER_LINEAR  = 1,
    BMCV_INTER_BICUBIC = 2
} bmcv_resize_algorithm;
```

参数说明

- BMCV_INTER_NEAREST

最近邻插值算法

- BMCV_INTER_LINEAR

双线性插值算法

- BMCV_INTER_BICUBIC

双三次插值算法

4.8 Format

定义常用的图像格式。

接口形式:

```
FORMAT_YUV420P
FORMAT_YUV422P
FORMAT_YUV444P
FORMAT_NV12
FORMAT_NV21
FORMAT_NV16
FORMAT_NV61
FORMAT_NV24
FORMAT_RGB_PLANAR
FORMAT_BGR_PLANAR
FORMAT_RGB_PACKED
FORMAT_BGR_PACKED
FORMAT_RGBP_SEPARATE
FORMAT_BGRP_SEPARATE
FORMAT_GRAY
FORMAT_COMPRESSED
```

参数说明:

- FORMAT_YUV420P

表示预创建一个 YUV420 格式的图片，有三个 plane

- FORMAT_YUV422P

表示预创建一个 YUV422 格式的图片，有三个 plane

- FORMAT_YUV444P

表示预创建一个 YUV444 格式的图片，有三个 plane

- FORMAT_NV12

表示预创建一个 NV12 格式的图片，有两个 plane

- FORMAT_NV21

表示预创建一个 NV21 格式的图片，有两个 plane

- FORMAT_NV16

表示预创建一个 NV16 格式的图片，有两个 plane

- FORMAT_NV61

表示预创建一个 NV61 格式的图片，有两个 plane

- FORMAT_RGB_PLANAR

表示预创建一个 RGB 格式的图片，RGB 分开排列, 有一个 plane

- FORMAT_BGR_PLANAR

表示预创建一个 BGR 格式的图片，BGR 分开排列, 有一个 plane

- FORMAT_RGB_PACKED

表示预创建一个 RGB 格式的图片，RGB 交错排列, 有一个 plane

- FORMAT_BGR_PACKED

表示预创建一个 BGR 格式的图片，BGR 交错排列, 有一个 plane

- FORMAT_RGBP_SEPARATE

表示预创建一个 RGB planar 格式的图片，RGB 分开排列并各占一个 plane，共有 3 个 plane

- FORMAT_BGRP_SEPARATE

表示预创建一个 BGR planar 格式的图片，BGR 分开排列并各占一个 plane，共有 3 个 plane

- FORMAT_GRAY

表示预创建一个灰度图格式的图片，有一个 plane

- FORMAT_COMPRESSED

表示预创建一个 VPU 内部压缩格式的图片，共有四个 plane，分别存放内容如下：

plane0: Y 压缩表

plane1: Y 压缩数据

plane2: CbCr 压缩表

plane3: CbCr 压缩数据

4.9 ImgDtype

定义几种图像的存储形式。

接口形式:

```
DATA_TYPE_EXT_FLOAT32
DATA_TYPE_EXT_1N_BYTE
DATA_TYPE_EXT_4N_BYTE
DATA_TYPE_EXT_1N_BYTE_SIGNED
DATA_TYPE_EXT_4N_BYTE_SIGNED
```

参数说明:

- DATA_TYPE_EXT_FLOAT32

表示图片的数据类型为 float32。

- DATA_TYPE_EXT_1N_BYTE

表示图片的数据类型为 uint8。

- DATA_TYPE_EXT_4N_BYTE

表示图片的数据类型为 uint8，且每 4 张图片的数据交错排列，数据读写效率更高。

- DATA_TYPE_EXT_1N_BYTE_SIGNED

表示图片的数据类型为 int8。

- DATA_TYPE_EXT_4N_BYTE_SIGNED

表示图片的数据类型为 int8，且每 4 张图片的数据交错排列，数据读写效率更高。

4.10 Tensor

Tensor 是模型推理的输入输出类型，包含了数据信息，实现内存管理。

4.10.1 构造函数 Tensor()

初始化 Tensor, 并为 Tensor 分配内存

接口形式:

```
Tensor(
    const std::vector<int>& shape={},
    bm_data_type_t          dtype=BM_FLOAT32);

Tensor(
    Handle                  handle,
    const std::vector<int>& shape,
    bm_data_type_t          dtype=BM_FLOAT32,
    bool                    own_sys_data,
    bool                    own_dev_data);
```

参数说明:

- handle: Handle

设备标识 Handle

- shape: std::vector<int>

设置 Tensor 的 shape

- dtype: Dtype

Tensor 的数据类型

- own_sys_data: bool

指示 Tensor 是否拥有 system memory

- own_dev_data: bool

指示 Tensor 是否拥有 device memory

示例代码:

```
#include "tensor.h"

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1,input_tensor2;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32; // dtype can choose  BM_FLOAT32,␣
↪BM_INT8, BM_UINT8, BM_INT32, BM_UINT32

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);
    input_tensor2 = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,␣
↪true, true);
```

(下页继续)

(续上页)

```
    return 0;
}
```

4.10.2 shape

获取 Tensor 的 shape

接口形式:

```
const std::vector<int>& shape() const;
```

返回值说明:

- tensor_shape : std::vector<int>

返回 Tensor 的 shape 的 vector。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // get shape
    std::vector<int> tensor_shape;
    tensor_shape = input_tensor1->shape();
    std::cout << "tensor shape: ";
    for(int i = 0; i < tensor_shape.size(); i++) {
        std::cout << tensor_shape[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

4.10.3 dtype

获取 Tensor 的数据类型

接口形式:

```
bm_data_type_t dtype() const;
```

返回值说明:

- data_type : bm_data_type_t

返回 Tensor 的数据类型。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // get dtype
    bm_data_type_t input_dtype_;
    input_dtype_ = input_tensor1->dtype();
    return 0;
}
```

4.10.4 scale_from

先对 data 按比例缩放，再将数据更新到 Tensor 的系统内存。

接口形式:

```
void scale_from(float* src, float scale, int size);
```

参数说明:

- src: float*

数据的起始地址

- scale: float32

等比例缩放时的尺度。

- size: int

数据的长度

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1,input_tensor2;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // prepare data
    std::shared_ptr<float> src_ptr(
            new float[10 * 10],
            std::default_delete<float[]>());
    float * src_data = src_ptr.get();
    for(int i = 0; i < 10 * 10; i++) {
        src_data[i] = rand() % 255;
    }

    // scale data len is 99
    input_tensor1->scale_from(src_data, 0.1, 99);

    return 0;
}
```

4.10.5 scale_to

先对 Tensor 进行等比例缩放，再将数据返回到系统内存。

接口形式:

```
void scale_to(float* dst, float scale);

void scale_to(float* dst, float scale, int size);
```

参数说明:

- dst: float*

数据的起始地址。

- scale: float32

等比例缩放时的尺度。

- size: int

数据的长度。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // prepare dst
    float* dst = new float[100];

    // scale data len is 99
    input_tensor1->scale_to(dst, 0.1, 99);

    // print scaled data
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;

    return 0;
}
```

4.10.6 reshape

对 Tensor 进行 reshape

接口形式:

```
void reshape(const std::vector<int>& shape);
```

参数说明:

- shape: std::vector<int>

设置期望得到的新 shape。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;
```

(下页继续)

(续上页)

```
    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // reshape from 10x10 to 2x50
    input_tensor1->reshape({2,50});

    // get shape
    std::vector<int> tensor_shape;
    tensor_shape = input_tensor1->shape();
    std::cout << "tensor new shape: ";
    for(int i = 0; i < tensor_shape.size(); i++) {
        std::cout << tensor_shape[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

4.10.7 own_sys_data

查询该 Tensor 是否拥有系统内存的数据指针。

接口形式:

```
bool& own_sys_data();
```

返回值说明:

- judge_ret: bool

如果拥有系统内存的数据指针则返回 True，否则 False。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,␣
↪false, true); // own sys mem:true, own dev mem:false

    // input_tensor: own sys or dev data
    bool _own_sys_data = input_tensor->own_sys_data();
    std::cout << "input_tensor own_sys_data:" << _own_sys_data << std::endl;
```

(下页继续)

(续上页)

```
    return 0;
}
```

4.10.8 own_dev_data

查询该 Tensor 是否拥有设备内存的数据

接口形式:

```
bool& own_dev_data();
```

返回值说明:

- judge_ret：bool

如果拥有设备内存中的数据则返回 True，否则 False。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,␣
↪true, false); // own sys mem:true, own dev mem:false

    // input_tensor: own sys or dev data
    bool _own_dev_data = input_tensor->own_dev_data();
    std::cout << "input_tensor own_dev_data:" << _own_dev_data << std::endl;

    return 0;
}
```

4.10.9 sync_s2d

将 Tensor 中的数据从系统内存拷贝到设备内存。

接口形式:

```
void sync_s2d();

void sync_s2d(int size);
```

参数说明:

- size: int

将特定 size 字节的数据从系统内存拷贝到设备内存。

接口形式:

```
void sync_s2d(Tensor* src, int offset_src, int offset_dst, int len);
```

参数说明:

- Tensor*: src

指定被拷贝的 Tensor。

- offset_src: int

指定被拷贝 Tensor 上的数据偏移几个元素后开始拷贝。

- offset_dst: int

指定拷贝目标 Tensor 上的数据偏移几个元素后开始拷贝。

- len: int

指定拷贝长度，既拷贝的元素个数。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // prepare data
    input_tensor->ones();

    // input_tensor -> sync_s2d(); // copy all data
    input_tensor -> sync_s2d(99); // copy part data

    // prepare another data: output_tensor, which is on sys mem, and don't have data
    // copy input_tensor to output_tensor
    std::shared_ptr<sail::Tensor> output_tensor;
    output_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,␣
↪true, true);

    sail::Tensor& input_ref = *input_tensor;
    output_tensor -> sync_s2d(input_ref,2,3,10);
```

(下页继续)

(续上页)

```
    // test if copy success
    // must copy to system memory and save to dst
    output_tensor -> sync_d2s();
    int size = 100;
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1, size);
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
    return 0;
}
```

4.10.10 sync_d2s

将 Tensor 中的数据从设备内存拷贝到系统内存。

接口形式:

```
void sync_d2s();

void sync_d2s(int size);
```

参数说明:

- size: int

将特定 size 字节的数据从设备内存拷贝到系统内存。

接口形式:

```
void sync_d2s(Tensor* src, int offset_src, int offset_dst, int len);
```

参数说明:

- Tensor*: src

指定被拷贝的 Tensor。

- offset_src: int

指定被拷贝 Tensor 上的数据偏移几个元素后开始拷贝。

- offset_dst: int

指定拷贝目标 Tensor 上的数据偏移几个元素后开始拷贝。

- len: int

指定拷贝长度，既拷贝的元素个数。

示例代码:

```cpp
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, false,
↪ true); // own sys mem:false, own dev mem:true

    // prepare data
    input_tensor->ones();

    input_tensor -> sync_d2s(); // copy all data
    // input_tensor -> sync_d2s(99); // copy part data

    // prepare another data: output_tensor, which is on sys mem, and don't have data
    // copy input_tensor to output_tensor
    std::shared_ptr<sail::Tensor> output_tensor;
    output_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,␣
↪true, true);

    sail::Tensor& input_ref = *input_tensor;
    output_tensor -> sync_d2s(input_ref,2,3,10);

    // test if copy success
    int size = 100;
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1, size);
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
    return 0;
}
```

4.10.11 sync_d2d

将另外一个 Tensor 设备内存上的数据拷贝到本 Tensor 的设备内存中。

接口形式:

```cpp
void sync_d2d(Tensor* src, int offset_src, int offset_dst, int len);
```

参数说明:

- Tensor*: src

指定被拷贝的 Tensor。

- offset_src: int

指定被拷贝 Tensor 上的数据偏移几个元素后开始拷贝。

- offset_dst: int

指定拷贝目标 Tensor 上的数据偏移几个元素后开始拷贝。

- len: int

指定拷贝长度，既拷贝的元素个数。

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    sail::Handle handle_(dev_id+1);
    std::shared_ptr<sail::Tensor> input_tensor,output_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, false,
↪ true); // on dev0
    output_tensor = std::make_shared<sail::Tensor>(handle_, input_shape, input_dtype,␣
↪false, true); // on dev1
    // prepare data
    input_tensor -> ones();

    // d2d
    sail::Tensor& input_ref = *input_tensor;
    output_tensor -> sync_d2d(input_ref,1,1,10);

    return 0;
}
```

4.10.12 dump_data

将 Tensor 中的数据写入到指定文件中

接口形式:

```
void dump_data(std::string file_name, bool bin = false);
```

参数说明:

- file_name: string

写入文件的路径

- bin: bool

是否采用二进制的形式存储 Tensor, 默认 false.

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;
    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // prepare data
    input_tensor->ones();

    input_tensor->dump_data("dumped_tensor.txt",false);
    input_tensor->dump_data("dumped_tensor_bin.bin",true);

    return 0;
}
```

4.10.13 memory_set

用 value 的前 N 个字节填充 Tensor 的内存，N 可以是 1、2、4，取决于 Tensor 的 dtype。

接口形式:

```
void memory_set(void* value);
```

参数说明:

- value: void*

需要填充的值。

示例代码:

```
void test_if_success(int size, std::shared_ptr<sail::Tensor> output_tensor){
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1);
    for (int i = 0; i < 100; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
}

int main() {
```

(下页继续)

(续上页)

```
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {3, 1920, 1080};
    bm_data_type_t input_dtype = BM_FLOAT32;
    input_tensor = std::make_shared<sail::Tensor>(handle,input_shape, input_dtype,true,
↪true);

    float src_data = 1.1;
    // memory set to tensor
    input_tensor->memory_set(src_data);
    test_if_success(3 * 1920 * 1080,input_tensor);

    return 0;
}
```

4.10.14 memory_set

将本 Tensor 的数据全部置为 c，在接口内部根据本 Tensor 的 dtype 对 c 做相应的类型转换，本接口可能会因为数据类型转换而带来精度损失，建议用上面的 memory_set 接口。

接口形式:

```
void memory_set(float c);
```

参数说明:

- c: float

需要填充的值。

示例代码:

```
void test_if_success(int size, std::shared_ptr<sail::Tensor> output_tensor){
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1);
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
}

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {1};
```

(下页继续)

(续上页)

```
    bm_data_type_t input_dtype = BM_FLOAT32;
    input_tensor = std::make_shared<sail::Tensor>(handle,input_shape, input_dtype,true,
↪true);

    float value_ = 1.1;
    input_tensor->memory_set(value_);
    test_if_success(1,input_tensor);

    return 0;
}
```

4.10.15 zeros

将本 Tensor 的数据全部置为 0。

接口形式:

```
void zeros();
```

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // prepare data
    input_tensor->zeros();

    return 0;
}
```

4.10.16 ones

将本 Tensor 的数据全部置为 1。

接口形式:

```
void ones();
```

示例代码:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // prepare data
    input_tensor->ones();

    return 0;
}
```

4.11 Engine

Engine 可以实现 bmodel 的加载与管理，是实现模型推理的主要模块。

4.11.1 构造函数

初始化 Engine

接口形式 1:

创建 Engine 实例，并不加载 bmodel

```
Engine(int tpu_id);

Engine(const Handle&   handle);
```

参数说明 1:

- tpu_id: int

指定 Engine 实例使用的智能视觉深度学习处理器的 id

- handle: Handle

指定 Engine 实例使用的设备标识 Handle

接口形式 2:

创建 Engine 实例并加载 bmodel，需指定 bmodel 路径或内存中的位置。

```
Engine(
  const std::string& bmodel_path,
  int                tpu_id,
```

(下页继续)

(续上页)

```
  IOMode             mode);

Engine(
  const std::string& bmodel_path,
  const Handle&      handle,
  IOMode             mode);

Engine(
   const void* bmodel_ptr,
   size_t      bmodel_size,
   int         tpu_id,
   IOMode      mode);

Engine(
  const void*        bmodel_ptr,
  size_t             bmodel_size,
  const Handle&      handle,
  IOMode             mode);
```

参数说明 2:

- bmodel_path: string

指定 bmodel 文件的路径

- tpu_id: int

指定 Engine 实例使用的智能视觉深度学习处理器的 id

- mode: IOMode

指定输入/输出 Tensor 所在的内存位置：系统内存或设备内存。

- bmodel_ptr: void*

bmodel 在系统内存中的起始地址。

- bmodel_size: size_t

bmodel 在内存中的字节数

示例代码:

```
#include "engine.h"

int main() {
   int dev_id = 0;
   sail::Handle handle(dev_id);
   sail::Engine engine1(dev_id);
   sail::Engine engine2(handle);
   std::string bmodel_path = "your_bmodel.bmodel";
   sail::Engine engine3(bmodel_path, dev_id, sail::SYSI);
   sail::Engine engine4(bmodel_path, handle, sail::SYSI);
```

(下页继续)

(续上页)

```
    // 打开文件输入流
    std::ifstream file(bmodel_path, std::ios::binary);
    // 获取文件大小
    file.seekg(0, std::ios::end);
    size_t bmodel_size = file.tellg();
    file.seekg(0, std::ios::beg);
    // 分配内存来存储模型数据
    char* bmodel_ptr = new char[bmodel_size];
    // 读取文件内容到内存中
    file.read(bmodel_ptr, bmodel_size);
    // 关闭文件输入流
    file.close();
    sail::Engine engine5(bmodel_ptr, bmodel_size, dev_id, sail::SYSI);
    delete [] bmodel_ptr;
    return 0;
}
```

4.11.2 get_handle

获取 Engine 中使用的设备句柄 sail.Handle

接口形式:

```
Handle get_handle();
```

返回值说明:

- handle: Handle

返回 Engine 中的设备句柄。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine1(dev_id);
    sail::Handle handle = engine1.get_handle();
    return 0;
}
```

4.11.3 load

将 bmodel 载入 Engine 中。

接口形式 1:

指定 bmodel 路径，从文件中载入 bmodel。

```
bool load(const std::string& bmodel_path);
```

参数说明 1:

- bmodel_path: string

bmodel 的文件路径

接口形式 2:

从系统内存中载入 bmodel。

```
bool load(const void* bmodel_ptr, size_t bmodel_size);
```

参数说明 2:

- bmodel_ptr: void*

bmodel 在系统内存中的起始地址。

- bmodel_size: size_t

bmodel 在内存中的字节数。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    if (!engine.load(bmodel_path)) {
        // load failed
        std::cout << "Engine load bmodel "<< bmodel_file << "failed" << "\n";
        exit(0);
    }
    return 0;
}
```

4.11.4 get_graph_names

获取 Engine 中所有载入的计算图的名称。

接口形式:

```
std::vector<std::string> get_graph_names();
```

返回值说明:

- graph_names: std::vector<std::string>

Engine 中所有计算图的 name 的数组。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    return 0;
}
```

4.11.5 set_io_mode

设置 Engine 的输入/输出 Tensor 所在的内存位置：系统内存或设备内存。

接口形式:

```
void set_io_mode(
    const std::string& graph_name,
    IOMode             mode);
```

参数说明:

- graph_name: string

需要配置的计算图的 name。

- mode: IOMode

设置 Engine 的输入/输出 Tensor 所在的内存位置：系统内存或设备内存。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
```

(下页继续)

(续上页)

```
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    engine.set_io_mode(bmodel_names[0], sail::SYSI);
    return 0;
}
```

4.11.6 get_input_names

获取选定计算图中所有输入 Tensor 的 name

接口形式:

```
std::vector<std::string> get_input_names(const std::string& graph_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

返回值说明:

- input_names: std::vector<std::string>

返回选定计算图中所有输入 Tensor 的 name 的列表。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    return 0;
}
```

4.11.7 get_output_names

获取选定计算图中所有输出 Tensor 的 name。

接口形式:

```
std::vector<std::string> get_output_names(const std::string& graph_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

返回值说明:

- output_names: std::vector<std::string>

返回选定计算图中所有输出 Tensor 的 name 的列表。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> output_names = engine.get_output_names(bmodel_names[0]);
    return 0;
}
```

4.11.8 get_max_input_shapes

查询选定计算图中所有输入 Tensor 对应的最大 shape。

在静态模型中，输入 Tensor 的 shape 是固定的，应等于最大 shape。

在动态模型中，输入 Tensor 的 shape 应小于等于最大 shape。

接口形式:

```
std::map<std::string, std::vector<int>> get_max_input_shapes(
    const std::string& graph_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

返回值说明:

- max_shapes: std::map<std::string, std::vector<int> >

返回输入 Tensor 中的最大 shape。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::map<std::string, std::vector<int>> input_max_shapes = engine.get_max_input_
↪shapes(bmodel_names[0]);
    return 0;
}
```

4.11.9 get_input_shape

查询选定计算图中特定输入 Tensor 的 shape。

接口形式:

```
std::vector<int> get_input_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: std::vector<int>

该 name 下的输入 Tensor 中的最大维度的 shape。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
```

(下页继续)

(续上页)

```
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    std::vector<int> input_shape_0 = engine.get_input_shape(bmodel_names[0],input_
↪names[0]);
    return 0;
}
```

4.11.10 get_max_output_shapes

查询选定计算图中所有输出 Tensor 对应的最大 shape。

在静态模型中，输出 Tensor 的 shape 是固定的，应等于最大 shape。

在动态模型中，输出 Tensor 的 shape 应小于等于最大 shape。

接口形式:

```
std::map<std::string, std::vector<int>> get_max_output_shapes(
    const std::string& graph_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

返回值说明:

- std::map<std::string, std::vector<int> >

返回输出 Tensor 中的最大 shape。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::map<std::string, std::vector<int>> output_max_shapes = engine.get_max_
↪output_shapes(bmodel_names[0]);
    return 0;
}
```

4.11.11 get_output_shape

查询选定计算图中特定输出 Tensor 的 shape。

接口形式:

```
std::vector<int> get_output_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: std::vector<int>

该 name 下的输出 Tensor 的 shape。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> output_names = engine.get_output_names(bmodel_names[0]);
    std::vector<int> output_shape_0 = engine.get_output_shape(bmodel_names[0],output_
↪names[0]);
    return 0;
}
```

4.11.12 get_input_dtype

获取特定计算图的特定输入 Tensor 的数据类型。

接口形式:

```
bm_data_type_t get_input_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- datatype: bm_data_type_t

返回 Tensor 中数据的数据类型。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    std::vector<int> input_dype_0 = engine.get_input_dtype(bmodel_names[0],input_
↪names[0]);
    return 0;
}
```

4.11.13 get_output_dtype

获取特定计算图的特定输出 Tensor 的数据类型。

接口形式:

```
bm_data_type_t get_output_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- datatype: bm_data_type_t

返回 Tensor 中数据的数据类型。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    std::vector<int> output_dype_0 = engine.get_output_dtype(bmodel_names[0],input_
↪names[0]);
    return 0;
}
```

4.11.14 get_input_scale

获取特定计算图的特定输入 Tensor 的 scale，只在 int8 模型中有效。

接口形式:

```
float get_input_scale(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- scale: float32

返回 Tensor 数据的 scale。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
```

（下页继续）

(续上页)

```
    float input_scale_0 = engine.get_input_scale(bmodel_names[0],input_names[0]);
    return 0;
}
```

4.11.15 get_output_scale

获取特定计算图的特定输出 Tensor 的 scale，只在 int8 模型中有效。

接口形式:

```
float get_output_scale(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- scale: float32

返回 Tensor 数据的 scale。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_name = engine.get_graph_names()[0];
    std::vector<std::string> input_names = engine.get_input_names(bmodel_name);
    float output_scale_0 = engine.get_output_scale(bmodel_names[0],input_names[0]);
    return 0;
}
```

4.11.16 process

在特定的计算图上进行前向推理。

接口形式:

```
void process(
  const std::string&              graph_name,
  std::map<std::string, Tensor*>& input,
  std::map<std::string, Tensor*>& output,
  std::vector<int>                core_list = {});

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,
  std::vector<int>                        core_list = {});
```

参数说明:

- graph_name: string

输入参数。特定的计算图 name。

- input: std::map<std::string, Tensor*>

输入参数。所有的输入 Tensor 的数据。

- input_shapes : std::map<std::string, std::vector<int> >

输入参数。所有传入 Tensor 的 shape。

- output: std::map<std::string, Tensor*>

输出参数。所有的输出 Tensor 的数据。

- core_list: std::vector<int>

输入参数。该参数仅对支持多核推理的处理器有效，可以选择推理时使用的 core。设 bmodel 为对应的核数为 N，若 core_list 为空则使用从 core0 开始的 N 个 core 做推理；若 core_list 的长度大于 N，则使用 core_list 中对应的前 N 个 core 做推理。对于仅支持单核推理的处理器可忽略此参数。

示例代码:

```
#include "engine.h"

int main() {
  int dev_id = 0;
  sail::Engine engine(dev_id);
  std::string bmodel_path = "your_bmodel.bmodel";
  engine.load(bmodel_path);
  std::vector<std::string> graph_names = engine.get_graph_names();
  std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);
```

(下页继续)

(续上页)

```
    std::vector<std::string> output_names = engine.get_input_names(graph_names[0]);

    std::vector<int> input_shape, output_shape;
    bm_data_type_t input_dtype, output_dtype;
    // allocate input and output tensors with both system and device memory
    // or you can use engine.create_input\output_tensors_map to create
    for (int i = 0; i < input_names.size(); i++) {
        input_shape = engine.get_input_shape(graph_names[0], input_names[i]);
        input_dtype = engine.get_input_dtype(graph_names[0], input_names[i]);
        input_tensor[i] = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,␣
↪true, true);
        input_tensors[input_names[i]] = input_tensor[i].get();
    }
    for (int i = 0; i < output_names.size(); i++) {
        output_shape = engine.get_output_shape(graph_names[0], output_names[i]);
        output_dtype = engine.get_output_dtype(graph_names[0], output_names[i]);
        output_tensor[i] = std::make_shared<sail::Tensor>(handle, output_shape[i], output_
↪dtype, true, true);
        output_tensors[output_names[i]] = output_tensor[i].get();
    }

    // process1
    engine.process(graph_names[0], input_tensors, output_tensors);

    // process2
    std::map<std::string, std::vector<int>> input_shapes;
    for (const auto& input_name : input_names) {
        input_shape = engine.get_input_shape(bmodel_names, input_name);
        input_shapes[input_name] = input_shape;
    }

    engine.process(graph_names[0], input_tensors, input_shapes, output_tensors);
    return 0;
}
```

4.11.17 get_device_id

获取 Engine 中的设备 id 号

接口形式:

```
int get_device_id() const;
```

返回值说明:

- tpu_id : int

返回 Engine 中的设备 id 号。

示例代码:

```cpp
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    int dev = engine.get_device_id();
    return 0;
}
```

4.11.18 create_input_tensors_map

创建输入 Tensor 的映射

接口形式:

```cpp
std::map<std::string, Tensor*> create_input_tensors_map(
    const std::string& graph_name,
    int create_mode = -1);
```

参数说明:

- graph_name: string

特定的计算图 name。

- create_mode: int

创建 Tensor 分配内存的模式。为 0 时只分配系统内存，为 1 时只分配设备内存，其他时则根据 Engine 中 IOMode 的配置分配。

返回值说明:

input: std::map<std::string, Tensor*>

返回字符串到 tensor 的映射。

示例代码:

```cpp
#include "engine.h"


int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    sail::Engine engine(bmodel_path, dev_id, sail::IOMode::SYSIO);
    std::string graph_name = engine.get_graph_names()[0];
    std::map<std::string, sail::Tensor> input_tensors_map = engine.create_input_tensors_
↪map(graph_name);
    return 0;
}
```

4.11.19 create_output_tensors_map

创建输入 Tensor 的映射，在 python 接口中为字典 dict{string : Tensor}

接口形式:

```
std::map<std::string, Tensor*> create_output_tensors_map(
    const std::string& graph_name,
    int create_mode = -1);
```

参数说明:

- graph_name: string

特定的计算图 name。

- create_mode: int

创建 Tensor 分配内存的模式。为 0 时只分配系统内存，为 1 时只分配设备内存，其他时则根据 Engine 中 IOMode 的配置分配。

返回值说明:

output: std::map<std::string, Tensor*>

返回字符串到 tensor 的映射。

示例代码:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    sail::Engine engine(bmodel_path, dev_id, sail::IOMode::SYSIO);
    std::string graph_name = engine.get_graph_names()[0];
    std::map<std::string, sail::Tensor> output_tensors_map = engine.create_output_
↪tensors_map(graph_name);
    return 0;
}
```

4.12 MultiEngine

多线程的推理引擎，实现特定计算图的多线程推理。

4.12.1 MultiEngine

初始化 MutiEngine。

接口形式:

```
MultiEngine(const std::string& bmodel_path,
        std::vector<int>   tpu_ids,
        bool               sys_out=true,
        int                graph_idx=0);
```

参数说明:

- bmodel_path: string

bmodel 所在的文件路径。

- tpu_ids: std::vector<int>

该 MultiEngine 可见的智能视觉深度学习处理器的 ID。

- sys_out: bool

表示是否将结果拷贝到系统内存，默认为 True

- graph_idx : int

特定的计算图的 index。

示例代码:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    return 0;
}
```

4.12.2 set_print_flag

设置是否打印调试信息。

接口形式:

```
void set_print_flag(bool print_flag);
```

参数说明:

- print_flag: bool

为 True 时，打印调试信息，否则不打印。

示例代码:

```cpp
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    engine.set_print_flag(true);
    return 0;
}
```

4.12.3 set_print_time

设置是否打印主要处理耗时。

接口形式:

```cpp
void set_print_time(bool print_flag);
```

参数说明:

- print_flag: bool

为 True 时，打印主要耗时，否则不打印。

示例代码:

```cpp
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    engine.set_print_time(true);
    return 0;
}
```

4.12.4 get_device_ids

获取 MultiEngine 中所有可用的智能视觉深度学习处理器的 id。

接口形式:

```cpp
std::vector<int> get_device_ids();
```

返回值说明:

- device_ids: std::vector<int>

返回可见的智能视觉深度学习处理器的 ids

示例代码:

```cpp
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<int> device_ids = engine.get_device_ids();
    return 0;
}
```

4.12.5 get_graph_names

获取 MultiEngine 中所有载入的计算图的名称。

接口形式:

```cpp
std::vector<std::string> get_graph_names();
```

返回值说明:

- graph_names: std::vector<std::string>

MultiEngine 中所有计算图的 name 的列表。

示例代码:

```cpp
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    return 0;
}
```

4.12.6 get_input_names

获取选定计算图中所有输入 Tensor 的 name

接口形式:

```cpp
std::vector<std::string> get_input_names(const std::string& graph_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

返回值说明:

- input_names: std::vector<std::string>

返回选定计算图中所有输入 Tensor 的 name 的列表。

示例代码:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);
    return 0;
}
```

4.12.7 get_output_names

获取选定计算图中所有输出 Tensor 的 name。

接口形式:

```
std::vector<std::string> get_output_names(const std::string& graph_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

返回值说明:

- output_names: std::vector<std::string>

返回选定计算图中所有输出 Tensor 的 name 的列表。

示例代码:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> output_name = engine.get_output_names(graph_names[0]);
    return 0;
}
```

4.12.8 get_input_shape

查询选定计算图中特定输入 Tensor 的 shape。

接口形式:

```
std::vector<int> get_input_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: std::vector<int>

该 name 下的输入 Tensor 中的最大维度的 shape。

示例代码:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);
    std::vector<int> input_shape = engine.get_input_shape(graph_names[0],input_
↪names[0]);
    return 0;
}
```

4.12.9 get_output_shape

查询选定计算图中特定输出 Tensor 的 shape。

接口形式:

```
std::vector<int> get_output_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

参数说明:

- graph_name: string

设定需要查询的计算图的 name。

- tensor_name: string

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: std::vector<int>

该 name 下的输出 Tensor 的 shape。

示例代码:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> output_names = engine.get_output_names(graph_names[0]);
    std::vector<int> output_shape = engine.get_output_shape(graph_names[0],output_
↪names[0]);
    return 0;
}
```

4.12.10 process

在特定的计算图上进行推理，需要提供系统内存的输入数据。

接口形式:

```
std::vector<std::map<std::string, Tensor*>> process(std::vector<std::map<std::string,␣
↪Tensor*>>& input_tensors);
```

参数说明:

- input_tensors: std::vector<std::map<std::string, Tensor*> >

输入的 Tensors。

返回值说明:

- output_tensors: std::vector<std::map<std::string, Tensor*> >

返回推理之后的结果。

示例代码:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};
```

(下页继续)

(续上页)

```
    std::string bmodel_path = "/home/jingyu/SAM-ViT-B_embedding_fp16_1b.bmodel";
    sail::MultiEngine engine(bmodel_path, dev_id);


    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);

    std::vector<int> input_shape = engine.get_input_shape(graph_names[0], input_
↪names[0]);

    // prepare one input tensor
    std::map<std::string, sail::Tensor*> input_tensors_map1;
    for (const auto& input_name : input_names) {
        sail::Tensor* input_tensor = new sail::Tensor(input_shape);
        input_tensors_map1[input_name] = input_tensor;
    }
    // prepare multi input...
    std::vector<std::map<std::string, sail::Tensor*>> input_tensors_vector;
    input_tensors_vector.push_back(input_tensors_map1);

    // get multi output
    auto output_tensors_vector = engine.process(input_tensors_vector);

    for(auto& pair : input_tensors_map1) {
        delete pair.second;
    }
    return 0;
}
```

4.13 bm_image

bm_image 是 BMCV 中的基本结构，封装了一张图像的主要信息，是后续 BMImage 和 BMImageArray 的内部元素。

结构体:

```
struct bm_image {
    int width;
    int height;
    bm_image_format_ext image_format;
    bm_data_format_ext data_type;
    bm_image_private* image_private;
};
```

bm_image 结构成员包括图片的宽高（width、height），图片格式 image_format，图片数据格式 data_type，以及该结构的私有数据。

示例代码:

```
#include "cvwrapper.h"

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_image.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg = decoder.read(handle);
    // Convert BMImage to bm_image data structure; here is a bm_image
    struct bm_image img = BMimg.data();

    // print image width, height, format and dtype
    std::cout << img.width << " " << img.height << " " << img.image_format << " " <
↪< img.data_type << std::endl;

    return 0;
}
```

4.14 BMImage

BMImage 封装了一张图片的全部信息，可利用 Bmcv 接口将 BMImage 转换为 Tensor 进行模型推理。

BMImage 也是通过 Bmcv 接口进行其他图像处理操作的基本数据类型。

4.14.1 构造函数 BMImage()

初始化 BMImage。

接口形式:

```
BMImage();

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

参数说明:

- handle: Handle

设定 BMImage 所在的设备句柄。

- h: int

图像的高。

- w: int

图像的宽。

- format：bm_image_format_ext

图像的格式。

- dtype: bm_image_data_format_ext

图像的数据类型。

4.14.2 width

获取图像的宽。

接口形式:

```
int width();
```

返回值说明:

- width：int

返回图像的宽。

4.14.3 height

获取图像的高。

接口形式:

```
int height();
```

返回值说明:

- height：int

返回图像的高。

4.14.4 format

获取图像的格式。

接口形式:

```
bm_image_format_ext format();
```

返回值说明:

- format：bm_image_format_ext

返回图像的格式。

4.14.5 dtype

获取图像的数据类型。

接口形式:

```
bm_image_data_format_ext dtype() const;
```

返回值说明:

- dtype: bm_image_data_format_ext

返回图像的数据类型。

4.14.6 data

获取 BMImage 内部的 bm_image。

接口形式:

```
bm_image& data();
```

返回值说明:

- img : bm_image

返回图像内部的 bm_image。

4.14.7 get_device_id

获取 BMImage 中的设备 id 号。

接口形式:

```
int get_device_id() const;
```

返回值说明:

- device_id : int

返回 BMImage 中的设备 id 号

4.14.8 get_handle

获取 BMImage 中的 Handle。

接口形式:

```
Handle& get_handle();
```

返回值说明:

- Handle：Handle

返回 BMImage 中的 Handle

4.14.9 get_plane_num

获取 BMImage 中图像 plane 的数量。

接口形式:

```
int get_plane_num() const;
```

返回值说明:

- planes_num：int

返回 BMImage 中图像 plane 的数量。

4.14.10 align

将 BMImage 64 对齐

接口形式:

```
int align();
```

返回值说明:

- ret：int

返回 BMImage 是否对齐成功,-1 代表失败,0 代表成功

4.14.11 check_align

获取 BMImage 中图像是否对齐

接口形式:

```
bool check_align()const;
```

返回值说明:

- ret：bool

1 代表已对齐,0 代表未对齐

4.14.12 unalign

将 BMImage 不对齐

接口形式:

```
int unalign();
```

返回值说明:

- ret：int

返回 BMImage 是否不对齐成功,-1 代表失败,0 代表成功

4.14.13 check_contiguous_memory

获取 BMImage 中图像内存是否连续

接口形式:

```
bool check_contiguous_memory()const;
```

返回值说明:

- ret：bool

1 代表连续,0 代表不连续

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_image.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg = decoder.read(handle);

    // Get the image information
    int width = BMimg.width();
    int height = BMimg.height();
    bm_image_format_ext format = BMimg.format();
    bm_image_data_format_ext dtype = BMimg.dtype();

    // Convert BMImage to bm_image data structure
    bm_image bmimg = BMimg.data();

    // Get the device id and handle
    int device_id = BMimg.get_device_id();
    sail::Handle handle_ = BMimg.get_handle();
    int plane_num = BMimg.get_plane_num();
```

(下页继续)

(续上页)

```
    std::cout << "Width: " << width << ", Height: " << height << ", Format: " <<␣
↪format << ", Data Type: " << dtype << ", Device ID: " << device_id << ", Plane␣
↪Num: " << plane_num << std::endl;

    int ret;
    // Align the image
    ret = BMimg.align();
    if (ret != 0) {
        std::cout << "Failed to align the image!" << std::endl;
    }
    std::cout << "is align: " << BMimg.check_align() << std::endl;

    // Unaling the image
    ret = BMimg.unalign();
    if (ret != 0) {
        std::cout << "Failed to unalign the image!" << std::endl;
    }
    std::cout << "is align: " << BMimg.check_align() << std::endl;

    // check contiguous memory
    std::cout << "is continues: " <<BMimg.check_contiguous_memory()<< std::endl;

    return 0;

}
```

4.15 BMImageArray

BMImageArray 是 BMImage 的数组，可为多张图片申请连续的内存空间。

在声明 BMImageArray 时需要根据图片数量指定不同的实例

例：4 张图片时 BMImageArray 的构造方式如：images = BMImageArray<4>()

4.15.1 构造函数

初始化 BMImageArray。

接口形式:

```
BMImageArray();

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

参数说明:

- handle: Handle

设定 BMImage 所在的设备句柄。

- h: int

图像的高。

- w: int

图像的宽。

- format : bm_image_format_ext

图像的格式。

- dtype: bm_image_data_format_ext

图像的数据类型。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "your_image.jpg"
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.width(),image.
↪height(),image.format(),image.dtype());

    return 0;
}
```

4.15.2 copy_from

将图像拷贝到特定的索引上。

接口形式:

```
int copy_from(int i, BMImage &data);
```

参数说明:

- i: int

输入需要拷贝到的 index

- data: BMImage

需要拷贝的图像数据。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "your_image.jpg"
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.width(),image.
↪height(),image.format(),image.dtype());
    // copy from BMImage
    int ret = images.copy_from(0,image);
    if (ret != 0) {
        std::cout << "copy_from failed" << std::endl;
        return -1;
    }
    return 0;
}
```

4.15.3 attach_from

将图像 attach 到特定的索引上，这里没有内存拷贝，所以需要原始数据已经被缓存。

接口形式:

```
int attach_from(int i, BMImage &data);
```

参数说明:

- i: int

输入需要拷贝到的 index

- data: BMImage

需要拷贝的图像数据。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "your_image.jpg"
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.width(),image.
↪height(),image.format(),image.dtype());
    // attach from BMImage
    ret = images.attach_from(1,image);
    if (ret != 0) {
        std::cout << "attach_from failed" << std::endl;
        return -1;
    }
    return 0;
}
```

4.15.4 get_device_id

获取 BMImageArray 中的设备号。

接口形式:

```
int get_device_id();
```

返回值说明:

- device_id: int

BMImageArray 中的设备 id 号

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "/data/jinyu.lu/jpu_test/1920x1080_yuvj420.jpg";
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.height(),image.
↪width(),image.format(),image.dtype());
```

(下页继续)

(续上页)

```
    // get devid
    int devid = images.get_device_id();
    std::cout << "devid: " << devid << std::endl;

    return 0;
}
```

4.16 Decoder

解码器，可实现图像或视频的解码。

图像解码像素格式支持说明:

- 硬解支持 jpeg baseline，其余支持软解；
- 视频支持硬解 h264,h265。输出的像素格式为 YUV-nv12、YUVJ420P 或者 YUV420P；

4.16.1 构造函数 Decoder()

初始化 Decoder。

接口形式:

```
Decoder(
    const std::string& file_path,
    bool               compressed = true,
    int                tpu_id = 0);
```

参数说明:

- file_path: str

图像或视频文件的 Path 或 RTSP 的 URL。

- compressed: bool

是否将解码的输出压缩为 NV12，default：True。开启之后可以节省内存、节省带宽，但是输入视频必须要满足宽能被 16 整除才行，且输入必须为视频时才能生效。

- tpu_id: int

设置使用的智能视觉深度学习处理器 id 号。

4.16.2 is_opened

判断源文件是否打开。

接口形式:

```
bool is_opened();
```

返回值说明:

- judge_ret: bool

打开成功返回 True，失败返回 False。

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    if (!decoder.is_opened()) {
        cout << "Failed to open the file!" << endl;
        return -1;
    }
    return 0;
}
```

4.16.3 read

从 Decoder 中读取一帧图像。

接口形式:

```
int read(Handle& handle, BMImage& image);
```

参数说明:

- handle: Handle

输入参数。Decoder 使用的智能视觉深度学习处理器的 Handle。

- image: BMImage

输出参数。将数据读取到 image 中。

返回值说明:

- judge_ret: int

读取成功返回 0，失败返回其他值。

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Handle handle(tpu_id);
    Decoder decoder(file_path, true, tpu_id);
    BMImage image;

    int ret = decoder.read(handle, image);
    if (ret != 0) {
        cout << "Failed to read a frame!" << endl;
        return ret;
    }
    return 0;
}
```

4.16.4 read_

从 Decoder 中读取一帧图像。

接口形式:

```
int read_(Handle& handle, bm_image& image);
```

参数说明:

- handle: Handle

输入参数。Decoder 使用的智能视觉深度学习处理器的 Handle。

- image: bm_image

输出参数。将数据读取到 image 中。

返回值说明:

- judge_ret: int

读取成功返回 0，失败返回其他值。

示例代码:

```
#include <sail/cvwrapper.h>
```

(下页继续)

(续上页)

```
using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Handle handle(tpu_id);
    Decoder decoder(file_path, true, tpu_id);
    BMImage image;
    bm_image bm_img = image.data();
    int ret = decoder.read_(handle, bm_img);
    if (ret != 0) {
        cout << "Failed to read a frame!" << endl;
        return ret;
    }

    return 0;
}
```

4.16.5 get_frame_shape

获取 Decoder 中 frame 中的 shape。

接口形式:

```
std::vector<int> get_frame_shape();
```

返回值说明:

- frame_shape: std::vector<int>

返回当前 frame 的 shape。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    vector<int> frame_shape = decoder.get_frame_shape();

    for (auto dim : frame_shape) {
        cout << dim << " ";
    }
```

(下页继续)

(续上页)

```
    cout << endl;

    return 0;
}
```

4.16.6 release

释放 Decoder 资源。

接口形式:

```
void release();
```

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    decoder.release();

    return 0;
}
```

4.16.7 reconnect

Decoder 再次连接。

接口形式:

```
int reconnect();
```

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;
```

(下页继续)

(续上页)

```
    Decoder decoder(file_path, true, tpu_id);
    if (decoder.reconnect() != 0) {
        cout << "Reconnect failed!" << endl;
        return -1;
    }

    return 0;
}
```

4.16.8 enable_dump

开启解码器的 dump 输入视频功能（不经编码），并缓存最多 1000 帧未解码的视频。

接口形式:

```
void enable_dump(int dump_max_seconds):
```

参数说明:

- dump_max_seconds: int

输入参数。dump 视频的最大时长，也是内部 AVpacket 缓存队列的最大长度。

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;
    int dump_max_seconds = 100;  // 假设要dump的最大时长为100秒

    Decoder decoder(file_path, true, tpu_id);
    decoder.enable_dump(dump_max_seconds);

    return 0;
}
```

4.16.9 disable_dump

关闭解码器的 dump 输入视频功能，并清空开启此功能时缓存的视频帧

接口形式:

```
void disable_dump():
    """ Disable  input video dump without encode.
    """
```

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    decoder.enable_dump(100);
    decoder.disable_dump();

    return 0;
}
```

4.16.10 dump

在调用此函数的时刻，dump 下前后数秒的输入视频。由于未经编码，必须 dump 下前后数秒内所有帧所依赖的关键帧。因而接口的 dump 实现以 gop 为单位，实际 dump 下的视频时长将高于输入参数时长。误差取决于输入视频的 gop_size，gop 越大，误差越大。

接口形式:

```
int dump(int dump_pre_seconds, int dump_post_seconds, std::string& file_path)
```

- dump_pre_seconds: int

输入参数。保存调用此接口时刻之前的数秒视频。

- dump_post_seconds: int

输入参数。保存调用此接口时刻之后的数秒视频。

- file_path: std::string&

输入参数。视频路径。

返回值说明:

- judge_ret: int

成功返回 0，失败返回其他值。

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    string output_file_path = "output_video_path.mp4";
    int tpu_id = 0;
    int dump_pre_seconds = 3;
    int dump_post_seconds = 3;

    Decoder decoder(file_path, true, tpu_id);
    int ret = decoder.dump(dump_pre_seconds, dump_post_seconds, output_file_path);
    if (ret != 0) {
        cout << "Dump failed with error code: " << ret << endl;
        return ret;
    }

    return 0;
}
```

4.17 Encoder

编码器，可实现图像或视频的编码，以及保存视频文件、推 rtsp/rtmp 流。

4.17.1 构造函数

初始化 Encoder。

图片编码器初始化：

图片编码器:

```
Encoder();
```

视频编码器初始化：

视频编码器接口形式 1:

```
Encoder(const std::string &output_path,
            Handle &handle,
            const std::string &enc_fmt,
            const std::string &pix_fmt,
            const std::string &enc_params,
```

(下页继续)

(续上页)

```
        int cache_buffer_length=5,
        int abort_policy=0);
```

视频编码器接口形式 2:

```
Encoder(const std::string &output_path,
        int device_id,
        const std::string &enc_fmt,
        const std::string &pix_fmt,
        const std::string &enc_params,
        int cache_buffer_length=5
        int abort_policy=0);
```

参数说明:

- output_path: string

输入参数。编码视频输出路径，支持本地文件（MP4，ts 等）和 rtsp/rtmp 流。

- handle: sail.Handle

输入参数。编码器 handle 实例。(与 device_id 二选一)

- device_id: int

输入参数。编码器 device_id。(与 handle 二选一，指定 device_id 时，编码器内部将会创建 Handle)

- enc_fmt: string

输入参数。编码格式，支持 h264_bm 和 h265_bm/hevc_bm。

- pix_fmt: string

输入参数。编码输出的像素格式，支持 NV12 和 I420。推荐使用 I420。

- enc_params: string

输入参数。编码参数，"width=1902:height=1080:gop=32:gop_preset=3:framerate=25:bitrate=2000"，其中 width 和 height 是必须的，默认用 bitrate 控制质量，单位为 kbps，参数中指定 qp 时 bitrate 失效。

- cache_buffer_length: int

输入参数。内部缓存队列长度，默认为 5。sail.Encoder 内部会维护一个缓存队列，从而在推流时提升流控容错。

- abort_policy: int

输入参数。缓存队列已满时，video_write 接口的拒绝策略。设为 0 时，video_write 接口立即返回-1。设为 1 时，pop 队列头。设为 2 时，清空队列。设为 3 时，阻塞直到编码线程消耗一帧，队列产生空位。

4.17.2 is_opened

判断编码器是否打开。

接口形式:

```
bool is_opened();
```

返回值说明:

- judge_ret: bool

编码器打开返回 True，失败返回 False。

示例代码:

```
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string out_path = "path/to/your/output/file";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);
    if(encoder.is_opened())
    {
        cout<<"succeed!"<<endl;
    }
    return 0;
}
```

4.17.3 pic_encode

编码一张图片，并返回编码后的 data。

接口形式 1:

```
int pic_encode(std::string& ext, bm_image &image, std::vector<u_char>& data);
```

接口形式 2:

```
int pic_encode(std::string& ext, BMImage &image, std::vector<u_char>& data);
```

参数说明:

- ext: string

输入参数。图片编码格式。".jpg" , ".png" 等。

- image: bm_image/BMImage

输入参数。输入图片，只支持 FORMAT_BGR_PACKED, DATA_TYPE_EXT_1N_BYTE 的图片。

- data: vector<u_char>

输入参数。字节向量, 用于保存编码后放在系统内存中的的数据。

返回值说明:

- size: int

编码后放在系统内存中的数据的有效长度。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_img_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);

    string out_path = "path/to/your/output/file";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);

    vector<u_char> data;
    string extension = ".jpg";
    int size = encoder.pic_encode(extension,img,data);
    //int size = encoder.pic_encode(extension,img.data(),data); //bm_image

    return 0;
}
```

4.17.4 video_write

向视频编码器送入一帧图像。异步接口，做格式转换后，放入内部的缓存队列中。

接口形式 1:

```
int video_write(bm_image &image);
```

接口形式 2:

```
int video_write(BMImage &image);
```

参数说明:

- image: bm_image/BMImage

在 BM1684 上，当编码器像素格式（即 pix_fmt）为 I420 时，待编码的 image 的 shape 可以与编码器的宽高不同；当像素格式为 NV12 时，要求 image 的 shape 与编码器的宽高一致，内部使用 bmcv_image_storage_convert 做格式转换，可能占用 NPU 资源。

在 BM1684X 上，待编码的 image 的 shape 可以与编码器的宽高不同，内部使用 bmcv_image_vpp_convert 做 resize 和格式转换。

返回值说明:

- judge_ret: int

成功返回 0，内部缓存队列已满返回-1。内部缓存队列中有一帧编码失败时返回-2。有一帧成功编码，但推流失败返回-3。未知的拒绝策略返回-4。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_img_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);
    string out_path = "out_put_path";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);
    int ret = encoder.video_write(img);
    // int ret = encoder.video_write(img.data());  //bm_image
```

（下页继续）

(续上页)

```
    return 0;
}
```

4.17.5 release

释放编码器。

示例代码:

```
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string out_path = "path/to/your/output/file";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);
    encoder.release();
    return 0;
}
```

4.18 Decoder_RawStream

裸流解码器，可实现 H264/H264 的解码。

4.18.1 构造函数 Decoder_RawStream()

初始化 Decoder_RawStream。

接口形式:

```
Decoder_RawStream(int tpu_id, string  decformt);
```

参数说明:

- tpu_id: int

设置智能视觉深度学习处理器的 id 号。

- decformat: string

输入参数。输入图像的格式，支持 h264 和 h265

4.18.2 read

从 Decoder_RawStream 中读取一帧图像。

接口形式:

```
int read(uint8_t* data, int data_size, sail::BMImage &image,bool continueFrame = false);
```

参数说明:

- data: uint8_t*

输入参数。裸流的二进制数据。

- image: BMImage

输出参数。将数据读取到 image 中。

- continueFrame: bool

输入参数。是否连续读帧, 默认为 false。

返回值说明:

- judge_ret: int

读取成功返回 0，失败返回其他值。

4.18.3 read_

从 Decoder_RawStream 中读取一帧图像。

接口形式:

```
int read_(uint8_t* data, int data_size, bm_image &image,bool continueFrame = false);
```

参数说明:

- data: uint8_t*

输入参数。裸流的二进制数据。

- image: bm_image

输出参数。将数据读取到 image 中。

- continueFrame: bool

输入参数。是否连续读帧, 默认为 false。

返回值说明:

- judge_ret: int

读取成功返回 0，失败返回其他值。

4.18.4 release

释放 Decoder 资源。

接口形式:

```
void release();
```

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;

int main(int argc, char *argv[]){

    const char *inputFile = "car.264";
    FILE *file = fopen(inputFile, "rb");
    if (!file) {
        fprintf(stderr, "Failed to open file for reading\n");
        return -1;
    }

    if(access("output",0)!=F_OK){
        mkdir("output",S_IRWXU);
    }

    fseek(file, 0, SEEK_END);
    int numBytes = ftell(file);
    cout << "infile size: " << numBytes << endl;
    fseek(file, 0, SEEK_SET);

    uint8_t *bs_buffer = (uint8_t *)av_malloc(numBytes);
    if (bs_buffer == nullptr) {
        cout << "av malloc for bs buffer failed" << endl;
        fclose(file);
        return -1;
    }

    fread(bs_buffer, sizeof(uint8_t), numBytes, file);
    fclose(file);
    file = nullptr;

    // create handle
    int dev_id=0;
    auto handle = sail::Handle(dev_id);
    bm_image  image;

    sail::Decoder_RawStream decoder_rawStream(dev_id,"h264");
```

(下页继续)

(续上页)

```
    int frameCount =0;
    while(true){
        decoder_rawStream.read_(bs_buffer,numBytes,image,true);
        string out = "output/out_" + to_string(frameCount) + ".bmp";
        bm_image_write_to_bmp(image, out.c_str());
        frameCount++;
    }
    av_free(bs_buffer);
    return 0;
}
```

4.19 Bmcv

Bmcv 封装了常用的图像处理接口，支持硬件加速。

4.19.1 构造函数 Bmcv()

初始化 Bmcv

接口形式:

```
Bmcv(Handle handle);
```

参数说明:

- handle: Handle

指定 Bmcv 使用的设备句柄。

4.19.2 bm_image_to_tensor

将 BMImage/BMImageArray 转换为 Tensor。

接口形式 1:

```
Tensor bm_image_to_tensor(BMImage &image);
```

参数说明 1:

- image: BMImage

需要转换的图像数据。

返回值说明 1:

- tensor: Tensor

返回转换后的 Tensor。

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_image_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor = bmcv.bm_image_to_tensor(img);
    return 0;
}
```

接口形式 2:

```
void bm_image_to_tensor(BMImage &img, Tensor &tensor);
```

参数说明 2:

- image: BMImageArray

输入参数。需要转换的图像数据。

- tensor: Tensor

输出参数。转换后的 Tensor。

示例代码:

```
#include <sail/cvwrapper.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_image_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor(handle,{1920,1080},BM_FLOAT32,true,true);
    bmcv.bm_image_to_tensor(img,tensor);
    return 0;
}
```

4.19.3 tensor_to_bm_image

将 Tensor 转换为 BMImage/BMImageArray。

接口形式 1:

```
void tensor_to_bm_image(Tensor &tensor, BMImage &img);

BMImage tensor_to_bm_image(Tensor &tensor);
```

参数说明 1:

- tensor: Tensor

输入参数。待转换的 Tensor。

- img：BMImage

转换后的图像。

返回值说明 1:

- image：BMImage

返回转换后的图像。

接口形式 2:

```
template<std::size_t N> void  bm_image_to_tensor (BMImageArray<N> &imgs, Tensor␣
↪&tensor);
template<std::size_t N> Tensor bm_image_to_tensor (BMImageArray<N> &imgs);
```

参数说明 2:

- tensor: Tensor

输入参数。待转换的 Tensor。

- img：BMImage | BMImageArray

输出参数。返回转换后的图像。

返回值说明 2:

- image：Tensor

返回转换后的 tensor。

示例代码 1:

```
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
```

(下页继续)

(续上页)

```
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor = bmcv.bm_image_to_tensor(BMimg);
    sail::BMImage BMimg2 = bmcv.tensor_to_bm_image(tensor);
    return 0;
    }
```

示例代码 2:

```
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor = bmcv.bm_image_to_tensor(BMimg);
    sail::BMImage new_img();
    bmcv.tensor_to_bm_image(tensor,new_img);
    return 0;
    }
```

4.19.4 crop_and_resize

对图片进行裁剪并 resize。

接口形式:

```
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,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

BMImage crop_and_resize(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
```

(下页继续)

(续上页)

```
  int                     resize_w,
  int                     resize_h,
  bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int crop_and_resize(
    BMImageArray<N>         &input,
    BMImageArray<N>         &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> crop_and_resize(
    BMImageArray<N>         &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

- resize_w：int

图像 resize 的目标宽度。

- resize_h : int

图像 resize 的目标高度。

- resize_alg : bmcv_resize_algorithm

图像 resize 的插值算法，默认为 bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output : BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.crop_and_resize(BMimg, 0, 0, BMimg.width(), BMimg.
↪height(), 640, 640);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    ssail::BMImage BMimg3;
    int ret = bmcv.crop_and_resize(BMimg, BMimg3,0, 0, BMimg.width(), BMimg.height(),␣
↪640, 640);
    return 0;
}
```

4.19.5 crop

对图像进行裁剪。

接口形式:

```
int crop(
  BMImage                &input,
  BMImage                &output,
  int                    crop_x0,
  int                    crop_y0,
  int                    crop_w,
  int                    crop_h);


BMImage crop(
  BMImage                &input,
  int                    crop_x0,
  int                    crop_y0,
  int                    crop_w,
  int                    crop_h);

template<std::size_t N>
int crop(
   BMImageArray<N>           &input,
   BMImageArray<N>           &output,
   int                       crop_x0,
   int                       crop_y0,
   int                       crop_w,
   int                       crop_h);

template<std::size_t N>
BMImageArray<N> crop(
   BMImageArray<N>           &input,
   int                       crop_x0,
   int                       crop_y0,
   int                       crop_w,
   int                       crop_h);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.crop(BMimg,100,100,200,200);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.crop(BMimg, BMimg3,100,100,200,200);
    return 0;
}
```

4.19.6 resize

对图像进行 resize。

接口形式:

```
int resize(
  BMImage                  &input,
  BMImage                  &output,
  int                      resize_w,
  int                      resize_h,
  bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

BMImage resize(
  BMImage                  &input,
  int                      resize_w,
  int                      resize_h,
  bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int resize(
   BMImageArray<N>          &input,
   BMImageArray<N>          &output,
   int                      resize_w,
   int                      resize_h,
   bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> resize(
   BMImageArray<N>          &input,
   int                      resize_w,
   int                      resize_h,
   bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- resize_alg：bmcv_resize_algorithm

图像 resize 的插值算法，默认为 bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.resize(BMimg,640,640);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.resize(BMimg, BMimg3,640,640);
    return 0;
}
```

4.19.7 vpp_crop_and_resize

利用 VPP 硬件加速图片的裁剪与 resize。

接口形式:

```
int vpp_crop_and_resize(
    BMImage                  &input,
    BMImage                  &output,
    int                    crop_x0,
    int                    crop_y0,
```

(下页继续)

(续上页)

```
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

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,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int vpp_crop_and_resize(
    BMImageArray<N>         &input,
    BMImageArray<N>         &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_crop_and_resize(
    BMImageArray<N>         &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0 : int

裁剪窗口在 y 轴上的起始点。

- crop_w : int

裁剪窗口的宽。

- crop_h : int

裁剪窗口的高。

- resize_w : int

图像 resize 的目标宽度。

- resize_h : int

图像 resize 的目标高度。

- resize_alg : bmcv_resize_algorithm

图像 resize 的插值算法，默认为 bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output : BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.vpp_crop_and_resize(BMimg,100,100,300,300,300,300);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
```

(下页继续)

(续上页)

```
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.vpp_crop_and_resize(BMimg, BMimg3,100,100,300,300,300,300);
    return 0;
}
```

4.19.8 vpp_crop_and_resize_padding

利用 VPP 硬件加速图片的裁剪与 resize，并 padding 到指定大小。

接口形式:

```
int vpp_crop_and_resize_padding(
    BMImage                 &input,
    BMImage                 &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_crop_and_resize_padding(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int vpp_crop_and_resize_padding(
    BMImageArray<N>         &input,
    BMImageArray<N>         &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);
```

(下页继续)

(续上页)

```
template<std::size_t N>
BMImageArray<N> vpp_crop_and_resize_padding(
    BMImageArray<N>           &input,
    int                       crop_x0,
    int                       crop_y0,
    int                       crop_w,
    int                       crop_h,
    int                       resize_w,
    int                       resize_h,
    PaddingAtrr               &padding_in,
    bmcv_resize_algorithm     resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- padding：PaddingAtrr

padding 的配置信息。

- resize_alg：bmcv_resize_algorithm

图像 resize 的插值算法，默认为 bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg4 = bmcv.vpp_crop_and_resize_padding(BMimg, 0, 0, BMimg.
↪width(), BMimg.height(), 640, 640, paddingatt);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    int ret = bmcv.vpp_crop_and_resize_padding(BMimg,BMimg3, 0, 0, BMimg.width(),␣
↪BMimg.height(), 640, 640, paddingatt);
    return 0;
```

（下页继续）

(续上页)

```
}
```

4.19.9 vpp_crop

利用 VPP 硬件加速图片的裁剪。

接口形式:

```
int vpp_crop(
  BMImage                 &input,
  BMImage                 &output,
  int                     crop_x0,
  int                     crop_y0,
  int                     crop_w,
  int                     crop_h);

BMImage vpp_crop(
  BMImage                 &input,
  int                     crop_x0,
  int                     crop_y0,
  int                     crop_w,
  int                     crop_h);

template<std::size_t N>
int vpp_crop(
   BMImageArray<N>          &input,
   BMImageArray<N>          &output,
   int                      crop_x0,
   int                      crop_y0,
   int                      crop_w,
   int                      crop_h);

template<std::size_t N>
BMImageArray<N> vpp_crop(
   BMImageArray<N>          &input,
   int                      crop_x0,
   int                      crop_y0,
   int                      crop_w,
   int                      crop_h);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0 : int

裁剪窗口在 y 轴上的起始点。

- crop_w : int

裁剪窗口的宽。

- crop_h : int

裁剪窗口的高。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output : BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.vpp_crop(BMimg,100,100,200,200);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.vpp_crop(BMimg, BMimg3,100,100,200,200);
    return 0;
}
```

4.19.10 vpp_resize

利用 VPP 硬件加速图片的 resize，采用最近邻插值算法。

接口形式 1:

```
int vpp_resize(
    BMImage                 &input,
    BMImage                 &output,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_resize(
    BMImage                 &input,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int vpp_resize(
    BMImageArray<N>          &input,
    BMImageArray<N>          &output,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_resize(
    BMImageArray<N>          &input,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- resize_alg：bmcv_resize_algorithm

图像 resize 的插值算法，默认为 bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.vpp_resize(BMimg,100,100,200,200);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.vpp_resize(BMimg, BMimg3,100,100,200,200);
    return 0;
}
```

4.19.11 vpp_resize_padding

利用 VPP 硬件加速图片的 resize，并 padding。

接口形式:

```
int vpp_resize_padding(
    BMImage                  &input,
    BMImage                  &output,
    int                    resize_w,
    int                    resize_h,
```

(下页继续)

(续上页)

```
    PaddingAtrr                &padding_in,
    bmcv_resize_algorithm      resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_resize_padding(
  BMImage                  &input,
  int                      resize_w,
  int                      resize_h,
  PaddingAtrr              &padding_in,
   bmcv_resize_algorithm      resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
   int vpp_resize_padding(
   BMImageArray<N>            &input,
   BMImageArray<N>            &output,
   int                      resize_w,
   int                      resize_h,
   PaddingAtrr              &padding_in,
   bmcv_resize_algorithm      resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_resize_padding(
   BMImageArray<N>            &input,
   int                      resize_w,
   int                      resize_h,
   PaddingAtrr              &padding_in,
   bmcv_resize_algorithm      resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- padding：PaddingAtrr

padding 的配置信息。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

- resize_alg：bmcv_resize_algorithm

图像 resize 的插值算法，默认为 bmcv_resize_algorithm.BMCV_INTER_NEAREST

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg4 = bmcv.vpp_resize_padding(BMimg, 0, 0, 640, 640, paddingatt);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    int ret = bmcv.vpp_resize_padding(BMimg,BMimg3, 640, 640, paddingatt);
    return 0;
}
```

4.19.12 warp

对图像进行仿射变换。

接口形式:

```
int warp(
  BMImage                       &input,
  BMImage                       &output,
  const std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>> &matrix,
  int                           use_bilinear = 0,
  bool                          similar_to_opencv = false);

BMImage warp(
  BMImage                       &input,
  const std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>> &matrix,
  int                           use_bilinear = 0,
  bool                          similar_to_opencv = false);

template<std::size_t N>
int warp(
    BMImageArray<N>                     &input,
    BMImageArray<N>                     &output,
    const std::array<
        std::pair<
        std::tuple<float, float, float>,
        std::tuple<float, float, float>>, N> &matrix,
    int                                 use_bilinear = 0,
    bool                                similar_to_opencv = false);

template<std::size_t N>
BMImageArray<N> warp(
    BMImageArray<N>                     &input,
    const std::array<
        std::pair<
        std::tuple<float, float, float>,
        std::tuple<float, float, float>>, N> &matrix,
    int                                 use_bilinear = 0,
    bool                                similar_to_opencv = false);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- output：BMImage | BMImageArray

处理后的图像或图像数组。

- **matrix: std::pair**< std::tuple<float, float, float>, std::tuple<float, float, float> >

2x3 的仿射变换矩阵。

- use_bilinear: int

是否使用双线性插值，默认为 0 使用最近邻插值，1 为双线性插值

- similar_to_opencv: bool

是否使用与 opencv 仿射变换对齐的接口

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
using AffineMatrix = std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>>;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    AffineMatrix rotated_matrix = std::make_pair(
        std::make_tuple(0.9996914396, -0.02484, 0.0f),
        std::make_tuple(0.02484, 0.9996914396, 0.0f)
    );
    sail::BMImage BMimg6 = bmcv.warp(BMimg, rotated_matrix);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
using AffineMatrix = std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>>;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
```

(下页继续)

(续上页)

```
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    AffineMatrix rotated_matrix = std::make_pair(
        std::make_tuple(0.9996914396, -0.02484, 0.0f),
        std::make_tuple(0.02484, 0.9996914396, 0.0f)
    );
    sail::BMImage BMimg6;
    int ret= bmcv.warp(BMimg,BMimg6, rotated_matrix);
    return 0;
}
```

4.19.13 convert_to

对图像进行线性变换。

接口形式:

```
int convert_to(
  BMImage                  &input,
  BMImage                  &output,
  const std::tuple<
    std::pair<float, float>,
    std::pair<float, float>,
    std::pair<float, float>>   &alpha_beta);

BMImage convert_to(
  BMImage                  &input,
  const std::tuple<
    std::pair<float, float>,
    std::pair<float, float>,
    std::pair<float, float>>   &alpha_beta);

template<std::size_t N>
int convert_to(
    BMImageArray<N>            &input,
    BMImageArray<N>            &output,
    const std::tuple<
        std::pair<float, float>,
        std::pair<float, float>,
        std::pair<float, float>>   &alpha_beta);

template<std::size_t N>
BMImageArray<N> convert_to(
    BMImageArray<N>            &input,
    const std::tuple<
        std::pair<float, float>,
        std::pair<float, float>,
        std::pair<float, float>>   &alpha_beta);
```

参数说明:

- input：BMImage | BMImageArray

待处理的图像或图像数组。

- **alpha_beta: std::tuple**< std::pair<float, float>, std::pair<float, float>, std::pair<float, float> >

分别为三个通道线性变换的系数 ((a0, b0), (a1, b1), (a2, b2))。

- output：BMImage | BMImageArray

输出参数。处理后的图像或图像数组。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::tuple<std::pair<float, float>, std::pair<float, float>, std::pair<float, float>> alpha_
↪beta =
        std::make_tuple(std::make_pair(1.0 / 255, 0), std::make_pair(1.0 / 255, 0), std::make_
↪pair(1.0 / 255, 0));
    sail::BMImage BMimg5 = bmcv.convert_to(BMimg, alpha_beta);
    return 0;
}
```

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
```

(下页继续)

(续上页)

```
    std::tuple<std::pair<float, float>, std::pair<float, float>, std::pair<float, float>> alpha_
↪beta =
        std::make_tuple(std::make_pair(1.0 / 255, 0), std::make_pair(1.0 / 255, 0), std::make_
↪pair(1.0 / 255, 0));
    sail::BMImage BMimg5;
    int ret = bmcv.convert_to(BMimg,BMimg5,alpha_beta);
    return 0;
}
```

4.19.14 yuv2bgr

将图像的格式从 YUV 转换为 BGR。

接口形式:

```
int yuv2bgr(
    BMImage                 &input,
    BMImage                 &output);

BMImage yuv2bgr(BMImage  &input);
```

参数说明:

- input：BMImage | BMImageArray

待转换的图像。

返回值说明:

- ret: int

返回 0 代表成功，其他代表失败。

- output：BMImage | BMImageArray

返回转换后的图像。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg5 = bmcv.yuv2bgr(BMimg);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg5;
    int ret = bmcv.yuv2bgr(BMimg,BMimg5);
    return 0;
}
```

4.19.15 rectangle

在图像上画一个矩形框。

接口形式:

```
int rectangle(
    BMImage                         &image,
    int                             x0,
    int                             y0,
    int                             w,
    int                             h,
    const std::tuple<int, int, int> &color,
    int                             thickness=1);
```

参数说明:

- image：BMImage

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

- thickness：int

矩形框线条的粗细。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.rectangle(BMimg, 20, 20, 600, 600, std::make_tuple(0, 0, 255), 2);
    return 0;
}
```

4.19.16 fillRectangle

在图像上画一个矩形。

接口形式:

```
int fillRectangle(
    BMImage                    &image,
    int                        x0,
    int                        y0,
    int                        w,
    int                        h,
    const std::tuple<int, int, int> &color);
```

参数说明:

- image：BMImage

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.fillRectangle(BMimg, 20, 20, 600, 600, std::make_tuple(0, 0, 255));
    return 0;
}
```

4.19.17 imwrite

将图像保存在特定文件。

接口形式:

```
int imwrite(
   const std::string &filename,
   BMImage           &image);
```

参数说明:

- file_name：string

文件的名称。

- output：BMImage

需要保存的图像。

返回值说明:

- process_status：int

如果保存成功返回 0，否则返回非 0 值。

示例代码:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.imwrite("new_3.jpg", BMimg);
    return 0;
}
```

4.19.18 get_handle

获取 Bmcv 中的设备句柄 Handle。

接口形式:

```cpp
Handle get_handle();
```

返回值说明:

- handle: Handle

Bmcv 中的设备句柄 Handle。

示例代码:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Handle handle1 = bmcv.get_handle();
    return 0;
}
```

4.19.19 crop_and_resize_padding

对图像进行裁剪并 resize，然后 padding。

接口形式:

```
bm_image crop_and_resize_padding(
    bm_image                &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

BMImage crop_and_resize_padding(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);
```

参数说明:

- input：BMImage/bm_image

待处理的图像。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- padding：PaddingAtrr

padding 的配置信息。

- resize_alg：bmcv_resize_algorithm

resize 采用的插值算法。

返回值说明:

- output：BMImage/bm_image

返回处理后的图像。

示例代码 1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg4 = bmcv.crop_and_resize_padding(BMimg, 0, 0, BMimg.width(),␣
↪BMimg.height(), 640, 640, paddingatt);
    return 0;
}
```

示例代码 2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
```

(下页继续)

(续上页)

```
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    bm_image bm_img = bmcv.crop_and_resize_padding(BMimg.data(), 0, 0, BMimg.
↪width(), BMimg.height(), 640, 640, paddingatt);
    return 0;
}
```

4.19.20 rectangle_

在图像上画一个矩形框。

接口形式:

```
int rectangle_(
  const bm_image              &image,
  int                         x0,
  int                         y0,
  int                         w,
  int                         h,
  const std::tuple<int, int, int> &color, // BGR
  int                         thickness=1);
```

参数说明:

- image：bm_image

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

- thickness：int

矩形框线条的粗细。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.rectangle_(BMimg, 20, 20, 600, 600, std::make_tuple(0, 0, 255), 2);
    return 0;
}
```

4.19.21 fillRectangle_

在图像上画一个矩形框。

接口形式:

```
int fillRectangle_(
    const bm_image                &image,
    int                           x0,
    int                           y0,
    int                           w,
    int                           h,
    const std::tuple<int, int, int> &color);
```

参数说明:

- image：bm_image

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.fillRectangle_(BMimg.data(), 20, 20, 600, 600, std::make_tuple(0, 0, 255));
    return 0;
}
```

4.19.22 imwrite_

将图像保存在指定的文件。

接口形式:

```
int imwrite_(
  const std::string &filename,
  const bm_image     &image);
```

参数说明:

- file_name：string

文件的名称。

- output：bm_image

需要保存的图像。

返回值说明:

- process_status：int

如果保存成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
```

(下页继续)

(续上页)

```
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.imwrite_("new_3.jpg", BMimg.data());
    return 0;
}
```

4.19.23 convert_format

将图像的格式转换为 output 中的格式，并拷贝到 output。

接口形式 1:

```
int convert_format(
    BMImage         &input,
    BMImage         &output
);
```

参数说明 1:

- input：BMImage

输入参数。待转换的图像。

- output：BMImage

输出参数。将 input 中的图像转化为 output 的图像格式并拷贝到 output。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4;
    int ret = bmcv.convert_format(BMimg,BMimg4);
    return 0;
}
```

接口形式 2:

将一张图像转换成目标格式。

```
BMImage convert_format(
    BMImage          &input,
    bm_image_format_ext image_format = FORMAT_BGR_PLANAR
);
```

参数说明 2:

- input：BMImage

待转换的图像。

- image_format：bm_image_format_ext

转换的目标格式。

返回值说明 2:

- output：BMImage

返回转换后的图像。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4 = bmcv.convert_format(BMimg);
    return 0;
}
```

4.19.24 vpp_convert_format

利用 VPP 硬件加速图片的格式转换。

接口形式 1:

```
int vpp_convert_format(
    BMImage          &input,
    BMImage          &output
);
```

参数说明 1:

- input：BMImage

输入参数。待转换的图像。

- output：BMImage

输出参数。将 input 中的图像转化为 output 的图像格式并拷贝到 output。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4;
    int ret = bmcv.vpp_convert_format(BMimg,BMimg4);
    return 0;
}
```

接口形式 2:

将一张图像转换成目标格式。

```
BMImage vpp_convert_format(
    BMImage          &input,
    bm_image_format_ext image_format = FORMAT_BGR_PLANAR
);
```

参数说明 2:

- input：BMImage

待转换的图像。

- image_format：bm_image_format_ext

转换的目标格式。

返回值说明 2:

- output：BMImage

返回转换后的图像。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4 = bmcv.vpp_convert_format(BMimg);
```

(下页继续)

(续上页)

```
    return 0;
}
```

4.19.25 putText

在图像上添加 text。

接口形式:

```
int putText(
    const BMImage                 &image,
    const std::string             &text,
    int                           x,
    int                           y,
    const std::tuple<int, int, int> &color, // BGR
    float                         fontScale,
    int                           thickness=1
);
```

参数说明:

- input：BMImage

待处理的图像。

- text: string

需要添加的文本。

- x: int

添加的起始点位置。

- y: int

添加的起始点位置。

- color：tuple

字体的颜色。

- fontScale: int

字号的大小。

- thickness：int

字体的粗细。

返回值说明:

- process_status：int

如果处理成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.putText(BMimg, "snow person" , 20, 20, std::make_tuple(0, 0, 255), 1.4,␣
↪2);
    return 0;
}
```

4.19.26 putText_

接口形式:

```
int putText_(
    const bm_image                  &image,
    const std::string               &text,
    int                             x,
    int                             y,
    const std::tuple<int, int, int> &color, // BGR
    float                           fontScale,
    int                             thickness=1
);
```

参数说明:

- input : bm_image

待处理的图像。

- text: string

需要添加的文本。

- x: int

添加的起始点位置。

- y: int

添加的起始点位置。

- color : tuple

字体的颜色。

- fontScale: int

字号的大小。

- thickness：int

字体的粗细。

返回值说明:

- process_status：int

如果处理成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.putText_(BMimg.data(), "snow person" , 20, 20, std::make_tuple(0, 0,␣
↪255), 1.4, 2);
    return 0;
}
```

4.19.27 image_add_weighted

将两张图像按不同的权重相加。

接口形式 1:

```
int image_add_weighted(
    BMImage         &input1,
    float           alpha,
    BMImage         &input2,
    float           beta,
    float           gamma,
    BMImage         &output
);
```

参数说明 1:

- input0：BMImage

输入参数。待处理的图像 0。

- alpha：float

输入参数。两张图像相加的权重 alpha

- input1：BMImage

输入参数。待处理的图像 1。

- beta：float

输入参数。两张图像相加的权重 beta

- gamma：float

输入参数。两张图像相加的权重 gamma

- output: BMImage

输出参数。相加后的图像 output = input1 * alpha + input2 * beta + gamma

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    float alpha=0.2,beta=0.5,gamma=0.8;
    int ret = bmcv.image_add_weighted(BMimg1,alpha,BMimg2,beta,gamma,BMimg2);
    return 0;
}
```

接口形式 2:

```
BMImage image_add_weighted(
    BMImage         &input1,
    float           alpha,
    BMImage         &input2,
    float           beta,
    float           gamma
);
```

参数说明 2:

- input0：BMImage

输入参数。待处理的图像 0。

- alpha：float

输入参数。两张图像相加的权重 alpha

- input1：BMImage

输入参数。待处理的图像 1。

- beta：float

输入参数。两张图像相加的权重 beta

- gamma : float

输入参数。两张图像相加的权重 gamma

返回值说明 2:

- output: BMImage

返回相加后的图像 output = input1 * alpha + input2 * beta + gamma

示例代码:

```
#include <sail/cvwrapper.h>
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    float alpha=0.2,beta=0.5,gamma=0.8;
    sail::BMImage img= bmcv.image_add_weighted(BMimg1,alpha,BMimg2,beta,gamma);
    return 0;
}
```

4.19.28 image_copy_to

进行图像间的数据拷贝

接口形式:

```
int image_copy_to(BMImage &input, BMImage &output, int start_x = 0, int start_y = 0);

template<std::size_t N>
int image_copy_to(BMImageArray<N> &input, BMImageArray<N> &output, int start_x
↪= 0, int start_y = 0);
```

参数说明:

- input: BMImage|BMImageArray

输入参数。待拷贝的 BMImage 或 BMImageArray。

- output: BMImage|BMImageArray

输出参数。拷贝后的 BMImage 或 BMImageArray

- start_x: int

输入参数。拷贝到目标图像的起始点。

- start_y: int

输入参数。拷贝到目标图像的起始点。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    bmcv.image_copy_to(BMimg1,BMimg2,0,0);
    return 0;
}
```

4.19.29 image_copy_to_padding

进行 input 和 output 间的图像数据拷贝并 padding。

接口形式:

```
int image_copy_to_padding(BMImage &input,
                          BMImage &output,
                          unsigned int padding_r,
                          unsigned int padding_g,
                          unsigned int padding_b,
                          int start_x = 0,
                          int start_y = 0);

template<std::size_t N>
int image_copy_to_padding(BMImageArray<N> &input,
                          BMImageArray<N> &output,
                          unsigned int padding_r,
                          unsigned int padding_g,
                          unsigned int padding_b,
                          int start_x = 0,
                          int start_y = 0);
```

参数说明:

- input: BMImage|BMImageArray

输入参数。待拷贝的 BMImage 或 BMImageArray。

- output: BMImage|BMImageArray

输出参数。拷贝后的 BMImage 或 BMImageArray

- padding_r: int

输入参数。R 通道的 padding 值。

- padding_g: int

输入参数。G 通道的 padding 值。

- padding_b: int

输入参数。B 通道的 padding 值。

- start_x: int

输入参数。拷贝到目标图像的起始点。

- start_y: int

输入参数。拷贝到目标图像的起始点。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    bmcv.image_copy_to_padding(BMimg1,BMimg2,128,128,128,0,0);
    return 0;
}
```

4.19.30 nms

利用智能视觉深度学习处理器进行 NMS

接口形式:

```
nms_proposal_t* nms(
    face_rect_t *input_proposal,
    int proposal_size,
    float threshold);
```

参数说明:

- input_proposal: face_rect_t

数据起始地址。

- proposal_size: int

待处理的检测框数据的大小。

- threshold: float

nms 的阈值。

返回值说明:

- result: nms_proposal_t

返回 NMS 后的检测框数组。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    face_rect_t *input_proposal;
    int proposal_size = 100;
    float threshold = 0.5;
    nms_proposal_t* result = bmcv.nms(input_proposal, proposal_size, threshold);
    return 0;
}
```

4.19.31 drawPoint

在图像上画点。

接口形式:

```
int drawPoint(
    const BMImage &image,
    std::pair<int,int> center,
    std::tuple<unsigned char, unsigned char, unsigned char> color,   // BGR
    int radius);
```

参数说明:

- image: BMImage

输入图像，在该 BMImage 上直接画点作为输出。

- center: std::pair<int,int>

点的中心坐标。

- color: std::tuple<unsigned char, unsigned char, unsigned char>

点的颜色。

- radius: int

点的半径。

返回值说明

如果画点成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path1";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.drawPoint(BMimg, std::pair(320, 320), std::make_tuple(0, 255, 255), 2);
    return 0;
}
```

4.19.32 drawPoint_

在图像上画点。

接口形式:

```
int drawPoint_(
    const bm_image  &image,
    std::pair<int,int> center,
    std::tuple<unsigned char, unsigned char, unsigned char> color,  // BGR
    int radius);
```

参数说明:

- image: bm_image

输入图像，在该 BMImage 上直接画点作为输出。

- center: std::pair<int,int>

点的中心坐标。

- color: std::tuple<unsigned char, unsigned char, unsigned char>

点的颜色。

- radius: int

点的半径。

返回值说明

如果画点成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.drawPoint(BMimg.data(), std::pair(320, 320), std::make_tuple(0, 255, 255),
↪ 2);
    return 0;
}
```

4.19.33 warp_perspective

对图像进行透视变换。

接口形式:

```
BMImage warp_perspective(
    BMImage                  &input,
    const std::tuple<
    std::pair<int,int>,
    std::pair<int,int>,
    std::pair<int,int>,
    std::pair<int,int>>      &coordinate,
    int                      output_width,
    int                      output_height,
    bm_image_format_ext      format = FORMAT_BGR_PLANAR,
    bm_image_data_format_ext dtype = DATA_TYPE_EXT_1N_BYTE,
    int                      use_bilinear = 0);
```

参数说明:

- input: BMImage

待处理的图像。

- **coordinate: std::tuple**< std::pair<int,int>, std::pair<int,int>, std::pair<int,int>, std::pair<int,int> >

变换区域的四顶点原始坐标。

例如 ((left_top.x, left_top.y), (right_top.x, right_top.y), (left_bottom.x, left_bottom.y), (right_bottom.x, right_bottom.y))

- output_width: int

输出图像的宽。

- output_height: int

输出图像的高。

- format: bm_image_format_ext

输出图像的格式。

- dtype: bm_image_data_format_ext

输出图像的数据类型。

- use_bilinear: int

是否使用双线性插值。

返回值说明:

- output: BMImage

输出变换后的图像。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;

int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::tuple<
        std::pair<int, int>,
        std::pair<int, int>,
        std::pair<int, int>,
        std::pair<int, int>
    > coordinate = std::make_tuple(
        std::make_pair(100, 100),
        std::make_pair(200, 100),
        std::make_pair(100, 200),
        std::make_pair(200, 200)
    )
    int output_width = 300;
    int output_height = 300;
    bm_image_format_ext format = FORMAT_BGR_PLANAR;
    bm_image_data_format_ext dtype = DATA_TYPE_EXT_1N_BYTE;
    int use_bilinear = 1;

    sail::BMImage output = bmcv.warp_perspective(BMimg,coordinate,output_width,
↪output_height,format,dtype,use_bilinear
    );
```

(下页继续)

(续上页)

```
    return 0;
}
```

4.19.34 get_bm_data_type

将 ImgDtype 转换为 Dtype

接口形式:

```
bm_data_type_t get_bm_data_type(bm_image_data_format_ext fmt);
```

参数说明:

- fmt: bm_image_data_format_ext

需要转换的类型。

返回值说明:

- ret: bm_data_type_t

转换后的类型。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    bm_data_type_t ret = bmcv.get_bm_data_type(bm_image_data_format_
↪ext::DATA_TYPE_EXT_FLOAT32);
    return 0;
}
```

4.19.35 get_bm_image_data_format

将 Dtype 转换为 ImgDtype。

接口形式:

```
bm_image_data_format_ext get_bm_image_data_format(bm_data_type_t dtype);
```

参数说明:

- dtype: bm_data_type_t

需要转换的 Dtype

返回值说明:

- ret: bm_image_data_format_ext

返回转换后的类型。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    bm_image_data_format_ext ret = bmcv.get_bm_image_data_format(bm_data_type_
↪t::BM_FLOAT32);
    return 0;
}
```

4.19.36 imdecode

从内存中载入图像到 BMImage 中。

接口形式:

```
BMImage imdecode(const void* data_ptr, size_t data_size);
```

参数说明:

- data_ptr: void*

数据起始地址

- data_size: bytes

数据长度

返回值说明:

- ret: BMImage

返回解码后的图像。

示例代码:

```
#include <sail/cvwrapper.h>
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    std::ifstream image_file(image_name, std::ios::binary);
    if (!image_file) {
```

(下页继续)

(续上页)

```
        std::cout << "Error opening image file." << std::endl;
        return -1;
    }
    std::vector<char> image_data_bytes(
        (std::istreambuf_iterator<char>(image_file)),
        (std::istreambuf_iterator<char>())
    );
    image_file.close();
    sail::Bmcv bmcv(handle);
    sail::BMImage src_img = bmcv.imdecode(image_data_bytes.data(), image_data_bytes.
↪size());
    return 0;
}
```

4.19.37 imencode

编码一张图片，并返回编码后的数据。

接口形式 1:

```
bool Bmcv::imencode(std::string& ext, bm_image &img, std::vector<u_char>& buf)
```

接口形式 2:

```
bool Bmcv::imencode(std::string& ext, BMImage &img, std::vector<u_char>& buf)
```

参数说明:

- ext: string

输入参数。图片编码格式。".jpg" , ".png" 等。

- image: bm_image/BMImage

输入参数。输入图片，只支持 FORMAT_BGR_PACKED，DATA_TYPE_EXT_1N_BYTE 的图片。

- buf: std::vector<u_char>

输出参数。编码后放在系统内存中的数据。

返回值说明:

- ret: bool

编码成功时返回 0，失败时返回 1。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
```

(下页继续)

(续上页)

```
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<u_char> encoded_data;
    std::string ext = ".jpg";
    bool success = bmcv.imencode(ext, BMimg, encoded_data);
    //bool success = bmcv.imencode(ext, BMimg.data(), encoded_data);  接口形式1:bm_
↪image
    return 0;
}
```

4.19.38 fft

实现对 Tensor 的快速傅里叶变换。

接口形式:

```
std::vector<Tensor> fft(bool forward, Tensor &input_real);

std::vector<Tensor> fft(bool forward, Tensor &input_real, Tensor &input_imag);
```

参数说明:

- forward: bool

是否进行正向迁移。

- input_real: Tensor

输入的实数部分。

- input_imag: Tensor

输入的虚数部分。

返回值说明:

- ret: std::vector<Tensor>

返回输出的实数部分和虚数部分。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
```

(下页继续)

(续上页)

```
    sail::Bmcv bmcv(handle);
    std::vector<int> shape = {1,1,512, 512};
    sail::Tensor input_real(shape);
    bool forward = true;
    //std::vector<sail::Tensor> result_real = bmcv.fft(forward, input_real);
    sail::Tensor input_imag(shape);
    std::vector<sail::Tensor> result_complex  = bmcv.fft(forward, input_real,input_imag);
    return 0;
}
```

4.19.39 convert_yuv420p_to_gray

将 YUV420P 格式的图片转为灰度图。

接口形式 1:

```
int convert_yuv420p_to_gray(BMImage& input, BMImage& output);
```

参数说明 1:

- input：BMImage

输入参数。待转换的图像。

- output：BMImage

输出参数。转换后的图像。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path1";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage img;
    int ret = bmcv.convert_yuv420p_to_gray(BMimg, img);
    return 0;
}
```

接口形式 2:

将 YUV420P 格式的图片转为灰度图。

```
int convert_yuv420p_to_gray_(bm_image& input, bm_image& output);
```

参数说明 2:

- input：bm_image

待转换的图像。

- output：bm_image

转换后的图像。

示例代码:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path1";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage img;
    int ret = bmcv.convert_yuv420p_to_gray_(BMimg.data(), img.data());
    return 0;
}
```

4.19.40 watermark_superpose

实现对图片添加多个水印。

接口形式:

```
int Bmcv::watermark_superpose(
BMImage &img,
string water_name,
int bitmap_type,
int pitch,
vector<vector<int>> rects,
vector<int> &color);
```

参数说明:

- image: BMImage

输入图片

- water_name:string

水印文件路径

- bitmap_type: int

输入参数。水印类型, 值 0 表示水印为 8bit 数据类型 (有透明度信息), 值 1 表示水印为 1bit 数据类型 (无透明度信息)。

- pitch: int

输入参数。水印文件每行的 byte 数, 可理解为水印的宽。

- rects: vector<vector<int🄵

输入参数。水印位置，包含每个水印起始点和宽高。

- color: vector<int>

输入参数。水印的颜色。

返回值说明:

- ret: int

返回是否成功

示例代码:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::string water_name = "your_watermark_path";
    int bitmap_type = 0;
    int pitch =117;
    std::vector<std::vector<int>> rects = {
        {10, 10, 117, 79},
        {200, 150, 117, 79}
    };
    std:: vector<int> color = {128,128,128};
    int result = bmcv.watermark_superpose(BMimg, water_name, bitmap_type, pitch, rects,␣
↪color);
    if (result == 0) {
        std::cout << "Watermarks added successfully." << std::endl;
    } else {
        std::cout << "Failed to add watermarks." << std::endl;
    }
    return 0;
}
```

4.19.41 polylines

可以实现在一张图像上画一条或多条线段，从而可以实现画多边形的功能，并支持指定线的颜色和线的宽度。

接口形式:

```
int polylines(
    BMImage &img,
    std::vector<std::vector<std::pair<int,int>>> &pts,
    bool isClosed,
    std::tuple<unsigned char, unsigned char, unsigned char> color,
    int thickness = 1,
    int shift = 0);
```

参数说明:

- img：BMImage

输入图片。

- pts：std::vector<std::vector<std::pair<int,int>>>

线段的起始点和终点坐标，可输入多个坐标点。图像左上角为原点，向右延伸为 x 方向，向下延伸为 y 方向。

- isClosed：bool

图形是否闭合。

- color：std::tuple<unsigned char, unsigned char, unsigned char>

画线的颜色，分别为 RGB 三个通道的值。

- thickness：int

画线的宽度，对于 YUV 格式的图像建议设置为偶数。

- shift：int

多边形缩放倍数，默认不缩放。缩放倍数为 (1/2)^shift。

返回值说明:

- ret: int

成功后返回 0

示例代码:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
```

(下页继续)

(续上页)

```
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<std::vector<std::pair<int, int>>> pts = {
        {{100, 100}, {150, 100}, {150, 150}, {100, 150}},
        {{200, 200}, {250, 200}, {250, 250}, {200, 250}}

    };
    bool isClosed = true;
    int thickness = 2;
    std::tuple<unsigned char, unsigned char, unsigned char> color = std::make_tuple(255, 0,␣
↪0);
    int shift = 0;
    int result = bmcv.polylines(BMimg, pts, isClosed, color, thickness, shift);
    if (result == 0) {
        std::cout << "Polylines drawn successfully." << std::endl;
    } else {
        std::cout << "Failed to draw polylines." << std::endl;
    }
    return 0;
}
```

4.19.42 mosaic

该接口用于在图像上打一个或多个马赛克。

接口形式:

```
int mosaic(
    int mosaic_num,
    BMImage &img,
    vector<vector<int>> rects,
    int is_expand);
```

参数说明:

- mosaic_num : int

马赛克数量，指 rects 中列表长度。

- img : BMImage

待转换的图像。

- rects : vector<vector<int

多个马赛克位置，列表中每个元素中参数为 [马赛克在 x 轴起始点, 马赛克在 y 轴起始点, 马赛克宽, 马赛克高]

- is_expand : int

是否扩列。值为 0 时表示不扩列, 值为 1 时表示在原马赛克周围扩列一个宏块 (8 个像素)。

返回值说明:

- ret: int

成功后返回 0

示例代码:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<std::vector<int>> rects = {
        {100, 100, 50, 50},
        {200, 200, 60, 60}
    };
    int mosaic_num = rects.size(); /
    int is_expand = 0;
    int result = bmcv.mosaic(mosaic_num, BMimg, rects, is_expand);
    if (result == 0) {
        std::cout << "Mosaic applied successfully." << std::endl;
    } else {
        std::.cout << "Failed to apply mosaic." << std::endl;
    }
    return 0;
}
```

4.19.43 gaussian_blur

该接口用于对图像进行高斯滤波。**注意：旧版本 SDK 并不支持 BM1684X，当前 SDK 是否支持请查询《BMCV 开发参考手册/BMCV API》查看。**

接口形式 1:

```
int gaussian_blur(
    BMImage                 &input,
    BMImage                 &output,
    int                     kw,
    int                     kh,
    float                   sigmaX,
    float                   sigmaY = 0.0f);
```

参数说明 1:

- input : BMImage

待转换的图像。

· output : BMImage

转换后输出的图像。

· kw : int

kernel 在 width 方向上的大小。

· kh : int

kernel 在 height 方向上的大小。

· sigmaX : float

X 方向上的高斯核标准差。

· sigmaY : float

Y 方向上的高斯核标准差。如果为 0 则表示与 X 方向上的高斯核标准差相同。默认为 0。

返回值说明 1:

· ret: int

成功后返回 0

接口形式 2: .. code-block:: c

BMImage gaussian_blur(BMImage &input, int kw, int kh, float sigmaX, float sigmaY = 0.0f);

参数说明 2: * input : BMImage

待转换的图像。

· kw : int

kernel 在 width 方向上的大小。

· kh : int

kernel 在 height 方向上的大小。

· sigmaX : float

X 方向上的高斯核标准差。

· sigmaY : float

Y 方向上的高斯核标准差。如果为 0 则表示与 X 方向上的高斯核标准差相同。默认为 0。

返回值说明 2:

· output: BMImage

返回经过高斯滤波的图像。

示例代码:

```cpp
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_img.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg_input = decoder.read(handle);
    sail::BMImage BMimg_output;
    sail::Bmcv bmcv(handle);
    int ret = bmcv.gaussian_blur(BMimg_input,BMimg_output,3, 3, 0.1);
    if(ret != 0){
        std::cout << "gaussian_blur failed" << std::endl;
        return -1;
    }
    bmcv.imwrite("output.jpg",BMimg_output);

    return 0;
}
```

4.19.44 transpose

该接口可以实现图片宽和高的转置。

接口形式 1:

```cpp
BMImage Bmcv::transpose(
    BMImage &src);
```

参数说明 1:

- src：BMImage

待转换的图像。

返回值说明 1:

- output: BMImage:

返回转换后的图像。

接口形式 2:

```cpp
int Bmcv::transpose(
    BMImage &src,
    BMImage &dst);
```

参数说明 2:

- src：BMImage

待转换的图像。

- dst：BMImage

输出图像的 BMImage 结构体。

返回值说明 2:

- ret：int

成功返回 0，否则返回非 0 值。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_img.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg_input = decoder.read(handle);
    sail::BMImage BMimg_output;
    sail::Bmcv bmcv(handle);
    int ret = bmcv.transpose(BMimg_input,BMimg_output);
    if(ret != 0){
        std::cout << "gaussian_blur failed" << std::endl;
        return -1;
    }
    bmcv.imwrite("output.jpg",BMimg_output);

    return 0;
}
```

4.19.45 Sobel

边缘检测 Sobel 算子。

注意：请查询《BMCV 开发参考手册/BMCV API》确认当前算子是否适配 BM1684X。

接口形式 1:

```
int Sobel(
    BMImage &input,
    BMImage &output,
    int dx,
    int dy,
    int ksize = 3,
    float scale = 1,
    float delta = 0);
```

参数说明 1:

- input：BMImage

待转换的图像。

- output：BMImage

转换后的图像。

- dx：int

x 方向上的差分阶数。

- dy：int

y 方向上的差分阶数。

- ksize：int

Sobel 核的大小，必须是-1,1,3,5 或 7。其中特殊的，如果是-1 则使用 3×3 Scharr 滤波器，如果是 1 则使用 3×1 或者 1×3 的核。默认值为 3。

- scale：float

对求出的差分结果乘以该系数，默认值为 1。

- delta：float

在输出最终结果之前加上该偏移量，默认值为 0。

返回值说明 1:

- ret: int

成功后返回 0

接口形式 2:

```
BMImage Sobel(
    BMImage &input,
    int dx,
    int dy,
    int ksize = 3,
    float scale = 1,
    float delta = 0);
```

参数说明 2:

- input：BMImage

待转换的图像。

- dx：int

x 方向上的差分阶数。

- dy：int

y 方向上的差分阶数。

- ksize：int

Sobel 核的大小，必须是-1,1,3,5 或 7。其中特殊的，如果是-1 则使用 3×3 Scharr 滤波器，如果是 1 则使用 3×1 或者 1×3 的核。默认值为 3。

- scale：float

对求出的差分结果乘以该系数，默认值为 1。

- delta：float

在输出最终结果之前加上该偏移量，默认值为 0。

返回值说明 2:

- output: BMImage

返回转换后的图像。

示例代码:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_img.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg_input = decoder.read(handle);
    sail::BMImage BMimg_output;
    sail::Bmcv bmcv(handle);
    int ret = bmcv.Sobel(BMimg_input,BMimg_output,1,1);
    if(ret != 0){
        std::cout << "Sobel failed" << std::endl;
        return -1;
    }
    bmcv.imwrite("output.jpg",BMimg_output);

    return 0;
}
```

4.20 MultiDecoder

多路解码接口，支持同时解码多路视频。

4.20.1 构造函数

接口形式:

```
MultiDecoder(
    int queue_size=10,
    int tpu_id=0,
    int discard_mode=0);
```

参数说明:

- queue_size: int

输入参数。每路视频，解码缓存图像队列的长度。

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id，默认为 0。

- discard_mode: int

输入参数。缓存达到最大值之后，数据的丢弃策略。0 表示不再放数据进缓存；1 表示先从队列中取出队列头的图片，丢弃之后再讲解码出来的图片缓存进去。默认为 0。

4.20.2 set_read_timeout

设置读取图片的超时时间，对 read 和 read_ 接口生效，超时之后仍然没有获取到图像，结果就会返回。

接口形式:

```
void set_read_timeout(int time_second);
```

参数说明:

- timeout: int

输入参数。超时时间，单位是秒。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    multiDecoder.set_read_timeout(100);
    return 0;
}
```

4.20.3 add_channel

添加一个通道。

接口形式:

```
int add_channel(
    const std::string&  file_path,
    int                 frame_skip_num=0);
```

参数说明:

- file_path: string

输入参数。视频的路径或者链接。

- frame_skip_num: int

输入参数。解码缓存的主动丢帧数，默认是 0，不主动丢帧。

返回值说明

返回视频对应的唯一的通道号。类型为整形。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    return 0;
}
```

4.20.4 del_channel

删除一个已经添加的视频通道。

接口形式:

```
int del_channel(int channel_idx);
```

参数说明:

- channel_idx: int

输入参数。将要删除视频的通道号。

返回值说明

成功返回 0，其他值时表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int ret = multiDecoder.del_channel(0);
    if (ret!=0) {
        cout << "Failed!" << endl;
        return -1;
    }
    return 0;
}
```

4.20.5 clear_queue

清除指定通道的图片缓存。

接口形式:

```
int clear_queue(int channel_idx);
```

参数说明:

- channel_idx: int

输入参数。将要删除视频的通道号。

返回值说明:

成功返回 0，其他值时表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int ret = multiDecoder.clear_queue(0);
    if (ret!=0) {
        cout << "Failed!" << endl;
        return -1;
    }
    return 0;
}
```

4.20.6 read

从指定的视频通道中获取一张图片。

接口形式 1:

```
int read(
    int         channel_idx,
    BMImage&    image,
    int         read_mode=0);
```

参数说明 1:

- channel_idx: int

输入参数。指定的视频通道号。

- image: BMImage

输出参数。解码出来的图片。

- read_mode: int

输入参数。获取图片的模式，0 表示不等待，直接从缓存中读取一张，无论有没有读取到都会返回。其他的表示等到获取到图片之后或等待时间超时再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }

    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            sail::BMImage bmimg;
```

(下页继续)

(续上页)

```
            int ret = multiDecoder.read(idx, bmimg, 1);
        }
        if (count == 20) {
            break;
        }
    }
    return 0;
}
```

接口形式 2:

```
BMImage read(int channel_idx);
```

参数说明 2:

- channel_idx: int

输入参数。指定的视频通道号。

返回值说明 2:

返回解码出来的图片，类型为 BMImage。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }

    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            sail::BMImage bmimg = multiDecoder.read(idx);
        }
        if (count == 20) {
            break;
        }
    }
```

(下页继续)

(续上页)

```
    return 0;
}
```

4.20.7 read_

从指定的视频通道中获取一张图片，通常是要和 BMImageArray 一起使用。

接口形式 1:

```
int read_(
    int         channel_idx,
    bm_image&   image,
    int         read_mode=0);
```

参数说明 1:

- channel_idx: int

输入参数。指定的视频通道号。

- image: bm_image

输出参数。解码出来的图片。

- read_mode: int

输入参数。获取图片的模式，0 表示不等待，直接从缓存中读取一张，无论有没有读取到都会返回。其他的表示等到获取到图片之后或等待时间超时再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
```

(下页继续)

(续上页)

```
    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            sail::BMImage image;
            bm_image bmimg = image.data()
            int ret = multiDecoder.read_(idx,bmimg,1);
        }
        if (count == 20) {
            break;
        }
    }
    return 0;
}
```

接口形式 2:

```
bm_image read_(int channel_idx);
```

参数说明 2:

- channel_idx: int

输入参数。指定的视频通道号。

返回值说明 2:

返回解码出来的图片，类型为 bm_image。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            bm_image bmimg =  multiDecoder.read_(idx);
```

(下页继续)

(续上页)

```
        }
        if (count == 20) {
            break;
        }
    }
    return 0;
}
```

4.20.8 reconnect

重连相应的通道的视频。

接口形式:

```
int reconnect(int channel_idx);
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

返回值说明

成功返回 0，其他值时表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>
using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int ret = multiDecoder.reconnect(0);
    if (ret!=0) {
        cout << "Failed!" << endl;
        return -1;
    }
    return 0;
}
```

4.20.9 get_frame_shape

获取相应通道的图像 shape。

接口形式:

```
std::vector<int> get_frame_shape(int channel_idx);
```

参数说明:

输入参数。输入图像的通道号。

返回值说明

返回一个由 1，通道数，图像高度，图像宽度组成的 list。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        vector<int> shape = multiDecoder.get_frame_shape(idx);
    }
    return 0;
}
```

4.20.10 set_local_flag

设置视频是否为本地视频。如果不调用则表示为视频为网络视频流。

接口形式:

```
void set_local_flag(bool flag);
```

参数说明:

- flag: bool

标准位，如果为 True，每路视频每秒固定解码 25 帧

示例代码:

```cpp
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>
using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    multiDecoder.set_local_flag(true);
    return 0;
}
```

4.20.11 get_channel_fps

Get the video fps of the specified channel

接口形式:

```cpp
float get_channel_fps(int channel_idx);
```

参数说明:

- channel_idx: int

指定需要获取视频帧数的视频通道号

返回值说明

返回指定视频通道的视频帧数

示例代码:

```cpp
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder  multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        float fps = multiDecoder.get_channel_fps(idx);
    }
    return 0;
}
```

get_drop_num

获取丢帧数。

接口形式:

```
size_t get_drop_num(int channel_idx);
```

参数说明:

输入参数。输入图像的通道号。

返回值说明

返回一个数代表丢帧数

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        size_t ret = multiDecoder.get_drop_num(idx);
    }
    return 0;
}
```

4.20.12 reset_drop_num

设置丢帧数为 0。

接口形式:

```
void reset_drop_num(int channel_idx);
```

参数说明:

输入参数。输入图像的通道号。

示例代码:

```cpp
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>
using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        multiDecoder.reset_drop_num(idx);
    }
    return 0;
}
```

4.21 sail_resize_type

图像预处理对应的预处理方法。

接口形式:

```cpp
enum sail_resize_type {
    BM_RESIZE_VPP_NEAREST = 0,
    BM_RESIZE_TPU_NEAREST = 1,
    BM_RESIZE_TPU_LINEAR = 2,
    BM_RESIZE_TPU_BICUBIC = 3,
    BM_PADDING_VPP_NEAREST = 4,
    BM_PADDING_TPU_NEAREST = 5,
    BM_PADDING_TPU_LINEAR = 6,
    BM_PADDING_TPU_BICUBIC = 7
};
```

参数说明:

- BM_RESIZE_VPP_NEAREST

使用 VPP，最近邻的方法进行图像尺度变换。

- BM_RESIZE_TPU_NEAREST

使用智能视觉深度学习处理器，最近邻的方法进行图像尺度变换。

- BM_RESIZE_TPU_LINEAR

使用智能视觉深度学习处理器，线性插值的方法进行图像尺度变换。

- BM_RESIZE_TPU_BICUBIC

使用智能视觉深度学习处理器，双三次插值的方法进行图像尺度变换。

- BM_PADDING_VPP_NEAREST

使用 VPP，最近邻的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_NEAREST

使用智能视觉深度学习处理器，最近邻的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_LINEAR

使用智能视觉深度学习处理器，线性插值的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_BICUBIC

使用智能视觉深度学习处理器，双三次插值的方法进行带 padding 的图像尺度变换。

4.22 ImagePreProcess

通用预处理接口，内部使用线程池的方式实现。

4.22.1 构造函数 ImagePreProcess()

接口形式:

```
ImagePreProcess(
    int batch_size,
    sail_resize_type resize_mode,
    int tpu_id=0,
    int queue_in_size=20,
    int queue_out_size=20,
    bool use_mat_flag=false);
```

参数说明:

- batch_size: int

输入参数。输出结果的 batch size。

- resize_mode: sail_resize_type

输入参数。内部尺度变换的方法。

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id，默认为 0。

- queue_in_size: int

输入参数。输入图像队列缓存的最大长度，默认为 20。

- queue_out_size: int

输入参数。输出 Tensor 队列缓存的最大长度，默认为 20。

- use_mat_output: bool

输入参数。是否使用 OpenCV 的 Mat 作为图片的输出，默认为 False，不使用。

4.22.2 SetResizeImageAtrr

设置图像尺度变换的属性。

接口形式:

```
void SetResizeImageAtrr(
    int output_width,
    int output_height,
    bool bgr2rgb,
    bm_image_data_format_ext  dtype);
```

参数说明:

- output_width: int

输入参数。尺度变换之后的图像宽度。

- output_height: int

输入参数。尺度变换之后的图像高度。

- bgr2rgb: bool

输入参数。是否将图像有 BGR 转换为 GRB。

- dtype: ImgDtype

输入参数。图像尺度变换之后的数据类型，当前版本只支持 BM_FLOAT32,BM_INT8,BM_UINT8。可根据模型的输入数据类型设置。

4.22.3 SetPaddingAtrr

设置 Padding 的属性，只有在 resize_mode 为 BM_PADDING_VPP_NEAREST、BM_PADDING_TPU_NEAREST、BM_PADDING_TPU_LINEAR、BM_PADDING_TPU_BICUBIC 时生效。

接口形式:

```
void SetPaddingAtrr(
    int padding_b=114,
    int padding_g=114,
    int padding_r=114,
    int align=0);
```

参数说明: * padding_b: int

输入参数。要 pdding 的 b 通道像素值，默认为 114。

- padding_g: int

输入参数。要 pdding 的 g 通道像素值，默认为 114。

- padding_r: int

输入参数。要 pdding 的 r 通道像素值，默认为 114。

- align: int

输入参数。图像填充为位置，0 表示从左上角开始填充，1 表示居中填充，默认为 0。

4.22.4 SetConvertAtrr

设置线性变换的属性。

接口形式:

```
int SetConvertAtrr(
  const std::tuple<
    std::pair<float, float>,
    std::pair<float, float>,
    std::pair<float, float>> &alpha_beta);
```

参数说明:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。输入参数。

  a0 描述了第 0 个 channel 进行线性变换的系数;

  b0 描述了第 0 个 channel 进行线性变换的偏移;

  a1 描述了第 1 个 channel 进行线性变换的系数;

  b1 描述了第 1 个 channel 进行线性变换的偏移;

  a2 描述了第 2 个 channel 进行线性变换的系数;

  b2 描述了第 2 个 channel 进行线性变换的偏移;

返回值说明:

设置成功返回 0，其他值时设置失败。

4.22.5 PushImage

送入数据。

接口形式:

```
int PushImage(
    int channel_idx,
    int image_idx,
    BMImage &image);
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- image: BMImage

输入参数。输入图像。

返回值说明:

设置成功返回 0，其他值时表示失败。

4.22.6 GetBatchData

获取处理的结果。

接口形式:

```
std::tuple<sail::Tensor,
    std::vector<BMImage>,
    std::vector<int>,
    std::vector<int>,
    std::vector<std::vector<int>>> GetBatchData();
```

返回值说明: tuple[data, images, channels, image_idxs, padding_attrs]

- data: Tensor

  处理后的结果 Tensor。

- images: std::vector<BMImage>

  原始图像序列。

- channels: std::vector<int>

  原始图像的通道序列。

- image_idxs: std::vector<int>

  原始图像的编号序列。

- padding_attrs: std::vector<std::vector<int> >

  填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度

4.22.7 set_print_flag

设置打印日志的标志位，不调用此接口时不打印日志。

接口形式:

```
void set_print_flag(bool print_flag);
```

返回值说明:

- flag: bool

打印的标志位，False 时表示不打印，True 时表示打印。

示例代码:

```
#include <sail/cvwrapper.h>
#include <opencv2/opencv.hpp>
#include <vector>

int main() {
    int tpu_id = 0;
    int batch_size = 1;
    std::string image_path = "./data/zidane.jpg";
    sail::Handle handle(tpu_id);

    std::vector<std::pair<int, int>> alpha_beta = {{1, 0}, {1, 0}, {1, 0}};
    sail::Decoder decoder(image_path, false, tpu_id);

    sail::ImagePreProcess sail_ipp(batch_size, sail::sail_resize_type::BM_RESIZE_VPP_
↪NEAREST, tpu_id, 20, 20, false);
    sail_ipp.SetResizeImageAtrr(640, 640, false, sail::ImgDtype::DATA_TYPE_EXT_1N_
↪BYTE);
    sail_ipp.SetConvertAtrr(alpha_beta);
    // sail_ipp.set_print_flag(true);
    sail::BMImage bm_i;
    for (int i = 0; i < batch_size; i++) {
        decoder.read(handle, bm_i);
        sail_ipp.PushImage(0, i, bm_i);
    }
    auto result = sail_ipp.GetBatchData();
    decoder.release();

    auto tensor = result[0];
    auto t_npy = tensor.asnumpy();
    auto result_img = t_npy[0].transpose({1, 2, 0});

    cv::Mat raw_img = cv::imread(image_path);
    cv::Mat resize_img = cv::resize(raw_img, cv::Size(640, 640), cv::INTER_NEAREST);
    double max_diff = abs((resize_img.astype(double) - result_img.astype(double)).max());
    double min_diff = abs((resize_img.astype(double) - result_img.astype(double)).min());
    double diff = std::max(max_diff, min_diff);
    std::cout << max_diff << " " << min_diff << " " << diff << std::endl;
```

（下页继续）

(续上页)

```
    return 0;
}
```

4.23 TensorPTRWithName

带有名称的 Tensor

```
struct TensorPTRWithName
{
    TensorPTRWithName(): name(""),data(NULL) { }
    std::string name;
    sail::Tensor* data;
};
```

参数说明

- name: string

tensor 的名字

- data: Tensor*

tensor 的数据

4.24 EngineImagePreProcess

带有预处理功能的图像推理接口，内部使用线程池的方式，Python 下面有更高的效率。

4.24.1 构造函数

接口形式:

```
EngineImagePreProcess(const std::string& bmodel_path,
                      int tpu_id,
                      bool use_mat_output=false,
                      std::vector<int> core_list = {});
```

参数说明: * bmodel_path: string

输入参数。输入模型的路径。

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id。

- use_mat_output: bool

输入参数。是否使用 OpenCV 的 Mat 作为图片的输出，默认为 False，不使用。

- use_mat_output: bool

输入参数。使用支持多核推理的处理器和 bmodel 时，可以选择推理时使用的多个核心，默认使用从 core0 开始的 N 个 core 来做推理，N 由当前 bmodel 决定。对于仅支持单核推理的处理器和 bmodel 模型，仅支持选择推理使用的单个核心，参数的输入列表长度必须为 1，若传入列表长度大于 1，将自动在 0 号核心上推理。默认为空不指定时，将默认从 0 号核心开始的 N 个 core 来做推理。

4.24.2 InitImagePreProcess

初始化图像预处理模块。

接口形式:

```
int InitImagePreProcess(
    sail_resize_type resize_mode,
    bool bgr2rgb=false,
    int queue_in_size=20,
    int queue_out_size=20);
```

参数说明:

- resize_mode: sail_resize_type

输入参数。内部尺度变换的方法。

- bgr2rgb: bool

输入参数。是否将图像有 BGR 转换为 GRB。

- queue_in_size: int

输入参数。输入图像队列缓存的最大长度，默认为 20。

- queue_out_size: int

输入参数。预处理结果 Tensor 队列缓存的最大长度，默认为 20。

返回值说明:

成功返回 0，其他值时失败。

4.24.3 SetPaddingAtrr

设置 Padding 的属性，只有在 resize_mode 为 BM_PADDING_VPP_NEAREST、BM_PADDING_TPU_NEAREST、BM_PADDING_TPU_LINEAR、BM_PADDING_TPU_BICUBIC 时生效。

接口形式:

```
int SetPaddingAtrr(
    int padding_b=114,
    int padding_g=114,
    int padding_r=114,
    int align=0);
```

参数说明: * padding_b: int

输入参数。要 pdding 的 b 通道像素值，默认为 114。

- padding_g: int

输入参数。要 pdding 的 g 通道像素值，默认为 114。

- padding_r: int

输入参数。要 pdding 的 r 通道像素值，默认为 114。

- align: int

输入参数。图像填充为位置，0 表示从左上角开始填充，1 表示居中填充，默认为 0。

返回值说明:

成功返回 0，其他值时失败。

4.24.4 SetConvertAtrr

设置线性变换的属性。

接口形式:

```
int SetConvertAtrr(
    const std::tuple<
        std::pair<float, float>,
        std::pair<float, float>,
        std::pair<float, float>> &alpha_beta);
```

参数说明:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。输入参数。

  a0 描述了第 0 个 channel 进行线性变换的系数;

  b0 描述了第 0 个 channel 进行线性变换的偏移;

  a1 描述了第 1 个 channel 进行线性变换的系数;

  b1 描述了第 1 个 channel 进行线性变换的偏移;

  a2 描述了第 2 个 channel 进行线性变换的系数;

  b2 描述了第 2 个 channel 进行线性变换的偏移;

返回值说明:

设置成功返回 0，其他值时设置失败。

4.24.5 PushImage

送入图像数据

接口形式:

```
int PushImage(
    int channel_idx,
    int image_idx,
    BMImage &image);
```

参数说明: * channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- image: BMImage

输入参数。输入的图像。

返回值说明:

成功返回 0，其他值时失败。

4.24.6 GetBatchData

获取一个 batch 的推理结果，调用此接口时，由于返回的结果类型为 BMImage，所以 use_mat_output 必须为 False。

接口形式:

```
std::tuple<std::map<std::string,sail::Tensor*>,
    std::vector<BMImage>,
    std::vector<int>,
    std::vector<int>,
    std::vector<std::vector<int>>> GetBatchData();
```

返回值说明:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: std::map<std::string,sail::Tensor*>

推理结果。

- ost_images: std::vector<BMImage>

原始图片序列。

- channels: std::vector<int>

结果对应的原始图片的通道序列。

- image_idxs: std::vector<int>

结果对应的原始图片的编号序列。

- padding_attrs: std::vector<std::vector<int> >

填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

4.24.7 GetBatchData_CV

获取一个 batch 的推理结果，调用此接口时，由于返回的结果类型为 cv::Mat，所以 use_mat_output 必须为 True。

接口形式:

```
std::tuple<std::map<std::string,sail::Tensor*>,
    std::vector<cv::Mat>,
    std::vector<int>,
    std::vector<int>,
    std::vector<std::vector<int>>> GetBatchData_CV();
```

返回值说明:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: std::map<std::string,sail::Tensor*>

推理结果。

- ost_images: std::vector<cv::Mat>

原始图片序列。

- channels: std::vector<int>

结果对应的原始图片的通道序列。

- image_idxs: std::vector<int>

结果对应的原始图片的编号序列。

- padding_attrs: std::vector<std::vector<int> >

填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

4.24.8 get_graph_name

获取模型的运算图名称。

接口形式:

```
std::string get_graph_name();
```

返回值说明:

返回模型的第一个运算图名称。

4.24.9 get_input_width

获取模型输入的宽度。

接口形式:

```
int get_input_width();
```

返回值说明:

返回模型输入的宽度。

4.24.10 get_input_height

获取模型输入的高度。

接口形式:

```
int get_input_height();
```

返回值说明:

返回模型输入的宽度。

4.24.11 get_output_names

获取模型输出 Tensor 的名称。

接口形式:

```
std::vector<std::string> get_output_names();
```

返回值说明:

返回模型所有输出 Tensor 的名称。

4.24.12 get_output_shape

获取指定输出 Tensor 的 shape

接口形式:

```
std::vector<int> get_output_shape(const std::string& tensor_name);
```

参数说明:

- tensor_name: string

指定的输出 Tensor 的名称。

返回值说明:

返回指定输出 Tensor 的 shape。

示例代码:

```
#include <sail/cvwrapper.h>
#include <opencv2/opencv.hpp>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_path = "./data/zidane.jpg";
    sail::Decoder decoder(image_path, true, dev_id);
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/
↪yolov5s_v6.1_3output_int8_1b.bmodel";
    std::vector<std::pair<float, float>> alpha_beta = {{1.0/255.0, 0}, {1.0/255.0, 0}, {1.0/
↪255.0, 0}};

    sail::sail_resize_type resize_type = sail::sail_resize_type::BM_PADDING_TPU_
↪LINEAR;
    sail::EngineImagePreProcess sail_engineipp(bmodel_path, dev_id, false);
    sail_engineipp.InitImagePreProcess(resize_type, false, 20, 20);
    sail_engineipp.SetPaddingAtrr();
    sail_engineipp.SetConvertAtrr(alpha_beta);

    int get_i_w = sail_engineipp.get_input_width();
    int get_i_h = sail_engineipp.get_input_height();
    std::string output_name = sail_engineipp.get_output_names()[0];
    std::vector<int> output_shape = sail_engineipp.get_output_shape(output_name);

    sail::BMImage bm_i;
    decoder.read(handle, bm_i);
```

(下页继续)

(续上页)

```
    sail_engineipp.PushImage(0, 0, bm_i);
    std::tuple<std::map<std::string,sail::Tensor*>,std::vector<BMImage>,std::vector<int>,
↪std::vector<int>,std::vector<std::vector<int>>> res = sail_engineipp.
↪GetBatchData(true);

    std::cout << output_name << " " << output_shape << " " << get_i_h << " " << ␣
↪get_i_w << " " << res << std::endl;
    return 0;
}
```

4.25 algo_yolov5_post_1output

针对以单输出 YOLOv5 模型的后处理接口，内部使用线程池的方式实现。

4.25.1 构造函数

接口形式:

```
algo_yolov5_post_1output(const std::vector<int>& shape,
                         int network_w=640,
                         int network_h=640,
                         int max_queue_size=20,
                         bool input_use_multiclass_nms=true,
                         bool agnostic=false);
```

参数说明:

- shape: std::vector<int>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

4.25.2 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

输入参数。输入图像序列的通道号。

- image_idx: std::vector<int>

输入参数。输入图像序列的编号。

- input_data: TensorPTRWithName

输入参数。输入数据。

- dete_threshold: std::vector<float>

输入参数。检测阈值序列。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- ost_w: std::vector<int>

输入参数。原始图片序列的宽。

- ost_h: std::vector<int>

输入参数。原始图片序列的高。

- padding_attrs: std::vector<std::vector<int> >

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

4.25.3 get_result_npy

获取最终的检测结果

接口形式:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
#include <stdio.h>
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
#include <sail/algokit.h>
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <queue>
#include <numeric>

int main() {
    int tpu_id = 0;
```

(下页继续)

(续上页)

```
    sail::Handle handle(tpu_id);
    std::string image_path = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/
↪yolov5s_v6.1_1output_int8_4b.bmodel";

    sail::Decoder decoder(image_path, true, tpu_id);
    sail::BMImage bmimg = decoder.read(handle);

    sail::EngineImagePreProcess engine_image_pre_process(bmodel_path, tpu_id, 0);
    engine_image_pre_process.PushImage(0, 0, bmimg);
    std::map<std::string,sail::Tensor*> output_tensor_map;
    std::vector<sail::BMImage> ost_images;
    std::vector<int> channel_list;
    std::vector<int> imageidx_list;
    std::vector<float> padding_atrr;
    engine_image_pre_process.GetBatchData(output_tensor_map, ost_images, channel_list,
↪ imageidx_list, padding_atrr);

    std::queue<std::vector<float>> post_queue;
    std::vector<int> width_list;
    std::vector<int> height_list;
    for (int index = 0; index < channel_list.size(); index++) {
        width_list.push_back(ost_images[index].width());
        height_list.push_back(ost_images[index].height());
    }
    post_queue.push(std::vector<float>({output_tensor_map, channel_list, imageidx_list,␣
↪width_list, height_list, padding_atrr}));

    sail::algo_yolov5_post_1output yolov5_post([4, 25200, 85], 640, 640, 10);
    std::vector<float> dete_thresholds(channels.size(), 0.2);
    std::vector<float> nms_thresholds(channels.size(), 0.5);
    yolov5_post.push_data(channel_list, imageidx_list, output_tensor_map, dete_
↪thresholds, nms_thresholds, width_list, height_list, padding_atrr);
    std::vector<std::tuple<int, int, int, int ,int, float>> objs;
    std::vector<int> channel;
    std::vector<int> image_idx;
    yolov5_post.get_result(&objs, &channel, &image_idx);
    std::cout << "objs: " << objs << ", channel: " << channel << ", image idx: " <<␣
↪image_idx << std::endl;

    return 0;
}
```

4.26 algo_yolov5_post_3output

针对以三输出 YOLOv5 模型的后处理接口，内部使用线程池的方式实现。

4.26.1 构造函数

接口形式:

```
algo_yolov5_post_3output(const std::vector<std::vector<int>>& shape,
                         int network_w=640,
                         int network_h=640,
                         int max_queue_size=20,
                         bool input_use_multiclass_nms=true,
                         bool agnostic=false);
```

参数说明:

- shape: std::vector<std::vector<int>>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

4.26.2 push_data

输入数据，支持任意 batchsize 的输入。

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    std::vector<TensorPTRWithName> input_data,
    std::vector<float> dete_threshold,
```

(下页继续)

(续上页)

```
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

输入参数。输入图像序列的通道号。

- image_idx: std::vector<int>

输入参数。输入图像序列的编号。

- input_data: std::vector<TensorPTRWithName>

输入参数。输入数据，包含三个输出。

- dete_threshold: std::vector<float>

输入参数。检测阈值序列。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- ost_w: std::vector<int>

输入参数。原始图片序列的宽。

- ost_h: std::vector<int>

输入参数。原始图片序列的高。

- padding_attrs: std::vector<std::vector<int🄵

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

4.26.3 get_result_npy

获取最终的检测结果

接口形式:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

4.26.4 reset_anchors

更新 anchor 尺寸.

接口形式:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

参数说明:

- anchors_new: std::vector<std::vector<std::vector<int> > >

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
#include <sail/algokit.h>
#include <vector>
#include <queue>
#include <numeric>
#include <iostream>
```

(下页继续)

(续上页)

```
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_path = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/
↪yolov5s_v6.1_3output_int8_1b.bmodel";

    sail::Decoder decoder(image_path, true, tpu_id);
    sail::BMImage bmimg = decoder.read(handle);

    sail::EngineImagePreProcess engine_image_pre_process(bmodel_path, tpu_id, 0);
    engine_image_pre_process.InitImagePreProcess(sail::sail_resize_type::BM_PADDING_
↪TPU_LINEAR, true, 10, 10);
    engine_image_pre_process.SetPaddingAtrr(114, 114, 114, 1);
    std::vector<std::pair<float, float>> alpha_beta = {{1.0/255, 0}, {1.0/255, 0}, {1.0/255,␣
↪0}};
    engine_image_pre_process.SetConvertAtrr(alpha_beta);
    bool ret = engine_image_pre_process.PushImage(0, 0, bmimg);

    std::vector<sail::Tensor> output_tensor_map;
    std::vector<sail::Image> ost_images;
    std::vector<int> channels;
    std::vector<int> imageidxs;
    std::vector<float> padding_atrr;
    engine_image_pre_process.GetBatchData(true, output_tensor_map, ost_images,␣
↪channels, imageidxs, padding_atrr);

    std::vector<int> width_list;
    std::vector<int> height_list;
    for (int index = 0; index < channels.size(); index++) {
        width_list.push_back(ost_images[index].width());
        height_list.push_back(ost_images[index].height());
    }

    sail::algo_yolov5_post_3output yolov5_post({[1, 3, 20, 20, 85], [1, 3, 40, 40, 85], [1, 3, 80,␣
↪80, 85]], 640, 640, 10);
    std::vector<float> dete_thresholds(channels.size(), 0.2);
    std::vector<float> nms_thresholds(channels.size(), 0.5);
    yolov5_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds, nms_
↪thresholds, width_list, height_list, padding_atrr);
    std::vector<std::tuple<int, int, int, int ,int, float>> objs;
    std::vector<int> channel;
    std::vector<int> image_idx;
    yolov5_post.get_result_npy(&objs, &channel, &image_idx);
    std::cout << "objs: " << objs << ", channel: " << channel << ", image idx: " <<␣
↪image_idx << std::endl;

    return 0;
}
```

4.27 algo_yolov5_post_cpu_opt_async

在处理器上，针对 YOLOv5 模型被加速的后处理接口，内部使用线程池的方式实现。

4.27.1 构造函数

接口形式:

```
algo_yolov5_post_cpu_opt_async(const std::vector<std::vector<int>>& shape,
                    int network_w=640,
                    int network_h=640,
                    int max_queue_size=20,
                    bool use_multiclass_nms=true);
```

参数说明:

- shape: std::vector<std::vector<int>>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- use_multiclass_nms: bool

输入参数。是否使用多类 NMS，默认为使用。

4.27.2 push_data

输入数据，支持任意 batchsize 的输入。

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    std::vector<TensorPTRWithName> input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

输入参数。输入图像序列的通道号。

- image_idx: std::vector<int>

输入参数。输入图像序列的编号。

- input_data: std::vector<TensorPTRWithName>

输入参数。输入数据，包含三个输出。

- dete_threshold: std::vector<float>

输入参数。检测阈值序列。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- ost_w: std::vector<int>

输入参数。原始图片序列的宽。

- ost_h: std::vector<int>

输入参数。原始图片序列的高。

- padding_attrs: std::vector<std::vector<int

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

4.27.3 get_result_npy

获取最终的检测结果

接口形式:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

4.27.4 reset_anchors

更新 anchor 尺寸.

接口形式:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

参数说明:

- anchors_new: std::vector<std::vector<std::vector<int> > >

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

4.28 tpu_kernel_api_yolov5_detect_out

针对 3 输出的 yolov5 模型，使用智能视觉深度学习处理器 Kernel 对后处理进行加速，目前只支持 BM1684x，且 libsophon 的版本必须不低于 0.4.6（v23.03.01）。

4.28.1 构造函数

接口形式:

```
tpu_kernel_api_yolov5_detect_out(int device_id,
                                 const std::vector<std::vector<int>>& shapes,
                                 int network_w=640,
```

（下页继续）

(续上页)

```
                        int network_h=640,
                        std::string module_file = "/opt/sophon/libsophon-current/lib/
↪tpu_module/libbm1684x_kernel_module.so");
```

参数说明:

- device_id: int

输入参数。使用的设备编号。

- shape: std::vector<std::vector<int> >

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- module_file: string

输入参数。Kernel module 文件路径，默认为”/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

4.28.2 process

处理接口。

接口形式 1:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                        std::vector<TensorPTRWithName>& input,
                        float dete_threshold,
                        float nms_threshold,
                        bool release_input = false);
```

参数说明 1:

- input_data: std::vector<TensorPTRWithName>

输入参数。输入数据，包含三个输出。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

- release_input: bool

输入参数。释放输入的内存，默认为 false。

接口形式 2:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                            std::map<std::string, Tensor&>& input,
                            float dete_threshold,
                            float nms_threshold,
                            bool release_input = false);
```

参数说明 2:

- input_data: std::map<std::string, Tensor&>

输入参数。输入数据，包含三个输出。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

- release_input: bool

输入参数。释放输入的内存，默认为 false。

返回值说明:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

4.28.3 reset_anchors

更新 anchor 尺寸.

接口形式:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

参数说明:

- anchors_new: std::vector<std::vector<std::vector<int> > >

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/tpu_kernel_api.h>
#include <opencv2/opencv.hpp>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <math.h>

using namespace std;

void get_ratio(sail::BMImage& bmimg, int& tw, int& th, int& tx1, int& tx2, int& ty1, int&
↪ty2) {
    int img_w = bmimg.width();
    int img_h = bmimg.height();
    double r_w = 640.0 / img_w;
    double r_h = 640.0 / img_h;
    if (r_h > r_w) {
        tw = 640;
        th = static_cast<int>(r_w * img_h);
        tx1 = tx2 = 0;
        ty1 = static_cast<int>((640 - th) / 2);
        ty2 = 640 - th - ty1;
    } else {
        tw = static_cast<int>(r_h * img_w);
        th = 640;
        tx1 = static_cast<int>((640 - tw) / 2);
        tx2 = 640 - tw - tx1;
        ty1 = ty2 = 0;
    }
}

int main() {
```

(下页继续)

(续上页)

```
    int tpu_id = 0;
    std::string image_path = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    sail::Decoder decoder(image_path, true, tpu_id);
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/
↪yolov5s_v6.1_3output_int8_1b.bmodel";
    sail::Handle handle(tpu_id);
    std::vector<std::pair<float, float>> alpha_beta = {{1.0/255, 0}, {1.0/255, 0}, {1.0/255,␣
↪0}};

    sail::sail_resize_type resize_type = sail::sail_resize_type::BM_PADDING_TPU_
↪LINEAR;
    sail::EngineImagePreProcess sail_engineipp(bmodel_path, tpu_id, false);
    sail_engineipp.InitImagePreProcess(resize_type, true, 10, 10);
    sail_engineipp.SetPaddingAtrr(114, 114, 114, 1);
    bool ret1 = sail_engineipp.SetConvertAtrr(alpha_beta);

    sail::BMImage bm_i;
    decoder.read(handle, bm_i);
    decoder.release();
    int hw, ratio, txy;
    get_ratio(bm_i, hw, ratio, txy);
    bool ret3 = sail_engineipp.PushImage(0, 0, bm_i);

    std::map<std::string,sail::Tensor*> output_tensor_map ;
    std::vector<BMImage> ost_images ;
    std::vector<int> channel_list ;
    std::vector<int> imageidx_list ;
    std::vector<std::vector<int>> padding_atrr ;
    std::tuple<output_tensor_map, ost_images, channel_list ,imageidx_list, padding_atrr>
↪all_out = sail_engineipp.GetBatchData(true);

    std::vector<std::vector<int>> shapes = {{1, 255, 80, 80}, {1, 255, 40, 40}, {1, 255, 20,␣
↪20}};
    sail::tpu_kernel_api_yolov5_detect_out tpu_kernel_3o(0, shapes, 640, 640, "/opt/
↪sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so");
    std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> out_boxs = tpu_
↪kernel_3o.process(output_tensor_map, 0.5, 0.5);
    std::vector<std::vector<float, 6>> result;
    for (int bid = 0; bid < out_boxs[0].size(); bid++) {
        std::vector<float, 6> temp_bbox;
        temp_bbox[0] = out_boxs[0][bid].class_id;
        if (temp_bbox[0] == -1)continue;
        temp_bbox[1] = out_boxs[0][bid].score;
        temp_bbox[2] = (out_boxs[0][bid].width+ 0.5) / ratio;
        temp_bbox[3] =(out_boxs[0][bid].height+ 0.5) / ratio;
        float centerX = ((out_boxs[0][bid].left + out_boxs[0][bid].right) / 2 + 1 - tx1) / ratio -
↪ 1;
        float centerY = ((out_boxs[0][bid].top + out_boxs[0][bid].bottom) / 2 + 1 - ty1) /␣
↪ratio - 1;
        temp_bbox[4] = MAX(int(centerX - temp_bbox.width / 2), 0);
        temp_bbox[5] = MAX(int(centerY - temp_bbox.height / 2), 0);
```

(下页继续)

(续上页)

```
        result.push_back(temp_bbox);
        }
    }
```

4.29 tpu_kernel_api_yolov5_out_without_decode

针对 1 输出的 yolov5 模型，使用智能视觉深度学习处理器 Kernel 对后处理进行加速，目前只支持 BM1684x，且 libsophon 的版本必须不低于 0.4.6（v23.03.01）。

4.29.1 构造函数

接口形式:

```
tpu_kernel_api_yolov5_out_without_decode(int device_id,
                                const std::vector<int>& shapes,
                                int network_w=640,
                                int network_h=640,
                                std::string module_file = "/opt/sophon/libsophon-current/lib/
↪tpu_module/libbm1684x_kernel_module.so");
```

参数说明:

- device_id: int

输入参数。使用的设备编号。

- shape: std::vector<int>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- module_file: string

输入参数。Kernel module 文件路径，默认为”/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

4.29.2 process

处理接口。

接口形式 1:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                        TensorPTRWithName& input,
                        float dete_threshold,
                        float nms_threshold);
```

参数说明 1:

· input_data: TensorPTRWithName

输入参数。输入数据，包含一个输出。

· dete_threshold: float

输入参数。检测阈值。

· nms_threshold: float

输入参数。nms 阈值序列。

接口形式 2:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                        std::map<std::string, Tensor&>& input,
                        float dete_threshold,
                        float nms_threshold);
```

参数说明 2:

· input_data: std::map<std::string, Tensor&>

输入参数。输入数据，包含一个输出。

· dete_threshold: float

输入参数。检测阈值。

· nms_threshold: float

输入参数。nms 阈值序列。

返回值说明:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

· left: int

检测结果最左 x 坐标。

· top: int

检测结果最上 y 坐标。

· right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/tpu_kernel_api.h>
#include <opencv2/opencv.hpp>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <math.h>

using namespace std;

void get_ratio(sail::BMImage& bmimg, int& tw, int& th, int& tx1, int& tx2, int& ty1, int&␣
↪ty2) {
    int img_w = bmimg.width();
    int img_h = bmimg.height();
    double r_w = 640.0 / img_w;
    double r_h = 640.0 / img_h;
    if (r_h > r_w) {
        tw = 640;
        th = static_cast<int>(r_w * img_h);
        tx1 = tx2 = 0;
        ty1 = static_cast<int>((640 - th) / 2);
        ty2 = 640 - th - ty1;
    } else {
        tw = static_cast<int>(r_h * img_w);
        th = 640;
        tx1 = static_cast<int>((640 - tw) / 2);
        tx2 = 640 - tw - tx1;
        ty1 = ty2 = 0;
    }
}

int main() {
    int tpu_id = 0;
    std::string image_path = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    sail::Decoder decoder(image_path, true, tpu_id);
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/
↪yolov5s_v6.1_1output_int8_1b.bmodel";
```

(下页继续)

(续上页)

```
    sail::Handle handle(tpu_id);
    std::vector<std::pair<float, float>> alpha_beta = {{1.0/255, 0}, {1.0/255, 0}, {1.0/255,␣
↪0}};

    sail::sail_resize_type resize_type = sail::sail_resize_type::BM_PADDING_TPU_
↪LINEAR;
    sail::EngineImagePreProcess sail_engineipp(bmodel_path, tpu_id, false);
    sail_engineipp.InitImagePreProcess(resize_type, true, 10, 10);
    sail_engineipp.SetPaddingAtrr(114, 114, 114, 1);
    bool ret1 = sail_engineipp.SetConvertAtrr(alpha_beta);

    sail::BMImage bm_i;
    decoder.read(handle, bm_i);
    decoder.release();
    int hw, ratio, txy;
    get_ratio(bm_i, hw, ratio, txy);
    bool ret3 = sail_engineipp.PushImage(0, 0, bm_i);

    std::map<std::string,sail::Tensor*> output_tensor_map ;
    std::vector<BMImage> ost_images ;
    std::vector<int> channel_list ;
    std::vector<int> imageidx_list ;
    std::vector<std::vector<int>> padding_atrr ;
    std::tuple<output_tensor_map, ost_images, channel_list ,imageidx_list, padding_atrr>
↪all_out = sail_engineipp.GetBatchData(true);

    sail::tpu_kernel_api_yolov5_out_without_decode tpu_kernel_1o(0, std::vector<int>{1,
↪ 25200, 85}, 640, 640, "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so");
    std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> out_boxs = tpu_
↪kernel_1o->process(output_tensor_map, 0.5, 0.5);
    std::vector<std::vector<float, 6>> result;
    for (int bid = 0; bid < out_boxs[0].size(); bid++) {
        std::vector<float, 6> temp_bbox;
        temp_bbox[0] = out_boxs[0][bid].class_id;
        if (temp_bbox[0] == -1)continue;
        temp_bbox[1] = out_boxs[0][bid].score;
        temp_bbox[2] = (out_boxs[0][bid].width+ 0.5) / ratio;
        temp_bbox[3] =(out_boxs[0][bid].height+ 0.5) / ratio;
        float centerX = ((out_boxs[0][bid].left + out_boxs[0][bid].right) / 2 + 1 - tx1) / ratio -
↪ 1;
        float centerY = ((out_boxs[0][bid].top + out_boxs[0][bid].bottom) / 2 + 1 - ty1) /␣
↪ratio - 1;
        temp_bbox[4] = MAX(int(centerX - temp_bbox.width / 2), 0);
        temp_bbox[5] = MAX(int(centerY - temp_bbox.height / 2), 0);
        result.push_back(temp_bbox);
        }
    }
```

4.30 sort_tracker_controller

基于 SORT 算法对追踪目标进行匹配

4.30.1 构造函数

接口形式:

```
deepsort_tracker_controller(float max_iou_distance = 0.7, int max_age = 30, int n_init =␣
↪3);
```

参数说明:

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

4.30.2 process

处理接口。

接口形式:

```
std::vector<std::tuple<int, int, int, int, int, float, int>> process(const std::vector<std::tuple
↪<int, int, int, int ,int, float>>& detected_objects_short);
```

参数说明:

- detected_objects_short:std::vector<std::tuple<left, top, right, bottom, class_id, score[F]

输入参数。检测出的物体框。

返回值说明:

- tracked_objects: std::vector<std::tuple<left, top, right, bottom, class_id, score, track_id[F]

输出参数。被跟踪的物体。

4.31 deepsort_tracker_controller_async

DeepSORT 算法异步处理接口

4.31.1 构造函数

接口形式:

```
deepsort_tracker_controller(float max_iou_distance = 0.7, int max_age = 30, int n_init =␣
↪3,int input_queue_size = 10,int result_queue_size = 10);
```

参数说明:

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

- input_queue_size: int

输入参数。输入数据缓冲队列的长度。

- result_queue_size: int

输入参数。结果缓冲队列的长度。

4.31.2 push_data

异步处理接口, 将输入参数推送到内部的任务队列，与 get_result 配合使用

接口形式 1:

```
int push_data(const std::vector<std::tuple<int, int, int, int ,int, float>>& detected_
↪objects_short);
```

参数说明 1:

- detected_objects:

输入参数。检测出的物体框。

返回值说明 1:

int

成功返回 0, 失败返回其他。

返回值说明 1:

int

成功返回 0, 失败返回其他。

4.31.3 get_result

异步处理接口，获取待追踪目标的信息，与 push_data 配合使用

接口形式:

```
std::vector<std::tuple<int, int, int, int, int, float, int>> get_result_npy();
```

返回值说明:

- std::vector<std::tuple<int, int, int, int, int, float, int🄵

输出参数。被跟踪的物体。

4.32 deepsort_tracker_controller

针对 DeepSORT 算法，通过处理检测的结果和提取的特征，实现对目标的跟踪。

4.32.1 构造函数

接口形式:

```
deepsort_tracker_controller(float max_cosine_distance,
                            int nn_budget,
                            int k_feature_dim,
                            float max_iou_distance = 0.7,
                            int max_age = 30,
                            int n_init = 3);
```

参数说明:

- max_cosine_distance: float

输入参数。用于相似度计算的最大余弦距离阈值。

- nn_budget: int

输入参数。用于最近邻搜索的最大数量限制。

- k_feature_dim: int

输入参数。被检测的目标的特征维度。

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

4.32.2 process

处理接口。

接口形式 1:

```
int process(const vector<DeteObjRect>& detected_objects,
        vector<Tensor>& feature,
        vector<TrackObjRect>& tracked_objects);
```

参数说明 1:

- detected_objects: vector<DeteObjRect>

输入参数。检测出的物体框。

- feature: vector<Tensor>

输入参数。检测出的物体的特征。

- tracked_objects: vector<TrackObjRect>

输出参数。被跟踪的物体。

返回值说明 1:

int

成功返回 0，失败返回其他。

接口形式 2:

```
int process(const vector<DeteObjRect>& detected_objects,
        vector<vector<float>>& feature,
        vector<TrackObjRect>& tracked_objects);
```

参数说明 2:

- detected_objects: vector<DeteObjRect>

输入参数。检测出的物体框。

- feature: vector<float>

输入参数。检测出的物体的特征。

- tracked_objects: vector<TrackObjRect>

输出参数。被跟踪的物体。

返回值说明 2:

int

成功返回 0，失败返回其他。

示例代码:

```
// The example code relies on sophon-demo/sample/YOLOv5/cpp/yolov5_bmcv/yolov5.h␣
↪and sophon-demo/sample/DeepSORT/cpp/deepsort_bmcv/FeatureExtractor.h
#include <sail/cvwrapper.h>
#include "yolov5.h"
#include "FeatureExtractor.h"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>

using namespace std;

class YOLOv5Arg {
public:
    string bmodel;
    int dev_id;
    float conf_thresh;
    float nms_thresh;

    YOLOv5Arg(string bmodel, int dev_id, float conf_thresh, float nms_thresh) {
        this->bmodel = bmodel;
        this->dev_id = dev_id;
        this->conf_thresh = conf_thresh;
        this->nms_thresh = nms_thresh;
    }
};

int main() {
    string input = "data/test_car_person_1080P.mp4";
    string bmodel_detector = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel";
    string bmodel_extractor = "models/BM1684X/extractor_int8_1b.bmodel";
    int dev_id = 0;
    float conf = 0.4;
    float nms = 0.7;

    YOLOv5Arg yolov5_args(bmodel_detector, dev_id, conf, nms);
    YOLOv5 yolov5(yolov5_args);
    Extractor extractor(bmodel_extractor, dev_id);

    cv::VideoCapture cap(input);
    vector<cv::Mat> img_batch;

    sail::deepsort_tracker_controller dstc(0.2, 100, extractor.output_shape[1], 0.7, 70, 3);
```

(下页继续)

(续上页)

```
    vector<vector<float>> track_res_all_numpy;

    for (int i = 0; i < 15; i++) {
        cv::Mat img;
        cap.read(img);
        if (img.empty()) {
            break;
        }
        img_batch.push_back(img);
        vector<vector<float>> det = yolov5(img_batch);
        vector<cv::Rect> dets;
        for (auto& item : det) {
            int x1 = static_cast<int>(item[0]);
            int y1 = static_cast<int>(item[1]);
            int x2 = static_cast<int>(item[2]);
            int y2 = static_cast<int>(item[3]);
            cv::Rect roi(x1, y1, x2 - x1, y2 - y1);
            dets.push_back(roi);
        }
        vector<cv::Mat> im_crops;
        for (auto& roi : dets) {
            cv::Mat img_crop = img(roi);
            im_crops.push_back(img_crop);
        }
        vector<vector<float>> ext_results = extractor(im_crops);

        // The order of this API and the demo is inconsistent, and the class_id and score are␣
↪reversed
        for (auto& row : det) {
            swap(row[4], row[5]);
        }
        img_batch.clear();

        vector<tuple<int, int, int, int, int, float, int>> det_tuple;
        for (auto& row : det) {
            det_tuple.push_back(make_tuple(static_cast<int>(row[0]), static_cast<int>
↪(row[1]), static_cast<int>(row[2]), static_cast<int>(row[3]), static_cast<int>(row[4]),␣
↪row[5], static_cast<int>(row[6])));
        }

    }
    return 0;
}
```

4.33 deepsort_tracker_controller_async

DeepSORT 算法异步处理接口

4.33.1 构造函数

接口形式:

```
deepsort_tracker_controller(float max_cosine_distance,
                            int nn_budget,
                            int k_feature_dim,
                            float max_iou_distance = 0.7,
                            int max_age = 30,
                            int n_init = 3,
                            int queue_size = 10);
```

参数说明:

- max_cosine_distance: float

输入参数。用于相似度计算的最大余弦距离阈值。

- nn_budget: int

输入参数。用于最近邻搜索的最大数量限制。

- k_feature_dim: int

输入参数。被检测的目标的特征维度。

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

- queue_size: int

输入参数。结果缓冲队列的长度。

4.33.2 push_data

异步处理接口, 将输入参数推送到内部的任务队列，与 get_result 配合使用

接口形式 1:

```
int push_data(const vector<DeteObjRect>& detected_objects,
              vector<Tensor>& feature);
```

参数说明 1:

- detected_objects: vector<DeteObjRect>

输入参数。检测出的物体框。

- feature: vector<Tensor>

输入参数。检测出的物体的特征。

返回值说明 1:

int

成功返回 0, 失败返回其他。

接口形式 2:

```
int push_data(const vector<DeteObjRect>& detected_objects,
              vector<vector<float>>& feature);
```

参数说明 2:

- detected_objects: vector<DeteObjRect>

输入参数。检测出的物体框。

- feature: vector<float>

输入参数。检测出的物体的特征。

返回值说明 1:

int

成功返回 0, 失败返回其他。

4.33.3 get_result

异步处理接口，获取待追踪目标的信息，与 push_data 配合使用

接口形式:

```
vector<TrackObjRect> get_result();
```

返回值说明:

- vector<TrackObjRect>

输出参数。被跟踪的物体。

示例代码:

```
// The example code relies on sophon-demo/sample/YOLOv5/cpp/yolov5_bmcv/yolov5.h␣
↪and sophon-demo/sample/DeepSORT/cpp/deepsort_bmcv/FeatureExtractor.h
#include <sail/cvwrapper.h>
#include "yolov5.h"
#include "FeatureExtractor.h"
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>

using namespace std;

class YOLOv5Arg {
public:
    string bmodel;
    int dev_id;
    float conf_thresh;
    float nms_thresh;

    YOLOv5Arg(string bmodel, int dev_id, float conf_thresh, float nms_thresh) {
        this->bmodel = bmodel;
        this->dev_id = dev_id;
        this->conf_thresh = conf_thresh;
        this->nms_thresh = nms_thresh;
    }
};

int main() {
    string input = "data/test_car_person_1080P.mp4";
    string bmodel_detector = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel";
    string bmodel_extractor = "models/BM1684X/extractor_int8_1b.bmodel";
    int dev_id = 0;
    float conf = 0.4;
    float nms = 0.7;

    YOLOv5Arg yolov5_args(bmodel_detector, dev_id, conf, nms);
    YOLOv5 yolov5(yolov5_args);
    Extractor extractor(bmodel_extractor, dev_id);

    cv::VideoCapture cap(input);
    vector<cv::Mat> img_batch;

    sail::deepsort_tracker_controller_async dstc(0.2, 100, extractor.output_shape[1], 0.7, 70,␣
↪3);

    vector<vector<float>> track_res_all_numpy;

    for (int i = 0; i < 15; i++) {
```

(下页继续)

(续上页)

```
        cv::Mat img;
        cap.read(img);
        if (img.empty()) {
            break;
        }
        img_batch.push_back(img);
        vector<vector<float>> det = yolov5(img_batch);
        vector<cv::Rect> dets;
        for (auto& item : det) {
            int x1 = static_cast<int>(item[0]);
            int y1 = static_cast<int>(item[1]);
            int x2 = static_cast<int>(item[2]);
            int y2 = static_cast<int>(item[3]);
            cv::Rect roi(x1, y1, x2 - x1, y2 - y1);
            dets.push_back(roi);
        }
        vector<cv::Mat> im_crops;
        for (auto& roi : dets) {
            cv::Mat img_crop = img(roi);
            im_crops.push_back(img_crop);
        }
        vector<vector<float>> ext_results = extractor(im_crops);

        // The order of this API and the demo is inconsistent, and the class_id and score are␣
↪reversed
        for (auto& row : det) {
            swap(row[4], row[5]);
        }
        img_batch.clear();

        vector<tuple<int, int, int, int, int, float, int>> det_tuple;
        for (auto& row : det) {
            det_tuple.push_back(make_tuple(static_cast<int>(row[0]), static_cast<int>
↪(row[1]), static_cast<int>(row[2]), static_cast<int>(row[3]), static_cast<int>(row[4]),␣
↪row[5], static_cast<int>(row[6])));
        }

    }
    return 0;
}
```

4.34 bytetrack_tracker_controller

针对 ByteTrack 算法，通过处理检测的结果，实现对目标的跟踪。

4.34.1 __init__

接口形式:

```
bytetrack_tracker_controller(int frame_rate = 30,
                    int track_buffer = 30);
```

参数说明:

- frame_rate: int

输入参数。用于控制被追踪物体允许消失的最大帧数，数值越大则被追踪物体允许消失的最大帧数越大。

- track_buffer: int

输入参数。用于控制被追踪物体允许消失的最大帧数，数值越大则被追踪物体允许消失的最大帧数越大。

4.34.2 process

处理接口。

接口形式 1:

```
int process(const vector<DeteObjRect>& detected_objects,
      vector<TrackObjRect>& tracked_objects);
```

参数说明 1:

- detected_objects: vector<DeteObjRect>

输入参数。检测出的物体框。

- tracked_objects: vector<TrackObjRect>

输出参数。被跟踪的物体。

返回值说明:

int

成功返回 0，失败返回其他。

示例代码:

```cpp
#include <sail/cvwrapper.h>
#include "yolov5.h"// The example code relies on sophon-demo/sample/YOLOv5/cpp/
↪yolov5_bmcv/yolov5.h
#include <opencv2/opencv.hpp>
#include <vector>
#include <string>

using namespace std;
using namespace cv;

class YOLOv5Arg {
public:
    string bmodel;
    int dev_id;
    float conf_thresh;
    float nms_thresh;

    YOLOv5Arg(string bmodel, int dev_id, float conf_thresh, float nms_thresh) {
        this->bmodel = bmodel;
        this->dev_id = dev_id;
        this->conf_thresh = conf_thresh;
        this->nms_thresh = nms_thresh;
    }
};

int main() {
    string input = "datasets/test_car_person_1080P.mp4";
    string bmodel = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel";
    int dev_id = 0;
    float conf = 0.4;
    float nms = 0.7;
    YOLOv5Arg yolov5_args(bmodel, dev_id, conf, nms);
    YOLOv5 yolov5(yolov5_args);

    VideoCapture cap(input);
    vector<Mat> img_batch;
    BytetrackTrackerController btt;
    vector<vector<float>> track_res_all;

    for (int i = 0; i < 50; i++) {
        Mat img;
        cap.read(img);
        if (img.empty()) {
            break;
        }
        img_batch.push_back(img);
        vector<vector<float>> results = yolov5.process(img_batch);
        vector<vector<float>> det = results[0];
        for (auto& row : det) {
            swap(row[4], row[5]);
        }
```

(下页继续)

(续上页)

```
        img_batch.clear();
        vector<tuple<int, int, int, int, int, float, int>> det_tuple;
        for (auto& row : det) {
            det_tuple.push_back(make_tuple(static_cast<int>(row[0]), static_cast<int>
↪(row[1]), static_cast<int>(row[2]), static_cast<int>(row[3]), static_cast<int>(row[4]),␣
↪row[5], static_cast<int>(row[6])));
        }
        vector<vector<float>> track_res = btt.process(det_tuple);
        track_res_all.push_back(track_res);
    }
    cap.release();
    return 0;
}
```

4.35 algo_yolox_post

针对 YOLOX 模型的后处理接口，内部使用线程池的方式实现。

4.35.1 构造函数

接口形式:

```
algo_yolox_post(const std::vector<int>& shape,
                int network_w=640,
                int network_h=640,
                int max_queue_size=20);
```

参数说明:

- shape: std::vector<int>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

4.35.2 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

输入参数。输入图像序列的通道号。

- image_idx: std::vector<int>

输入参数。输入图像序列的编号。

- input_data: TensorPTRWithName

输入参数。输入数据。

- dete_threshold: std::vector<float>

输入参数。检测阈值序列。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- ost_w: std::vector<int>

输入参数。原始图片序列的宽。

- ost_h: std::vector<int>

输入参数。原始图片序列的高。

- padding_attrs: std::vector<std::vector<int> >

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

4.35.3 get_result_npy

获取最终的检测结果

接口形式:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
#include <stdio.h>
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
#include <sail/algokit.h>
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
```

(下页继续)

(续上页)

```
    std::string bmodel_name = "../../../sophon-demo/sample/YOLOX/models/BM1684X/
↪yolox_int8_1b.bmodel";

    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);

    sail::EngineImagePreProcess engine_image_pre_process(bmodel_name, tpu_id, 0);
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_
↪TPU_LINEAR, true, 10, 10);
    engine_image_pre_process.SetPaddingAtrr(114, 114, 114, 1);
    std::vector<std::pair<float, float>> alpha_beta = {{1.0/255, 0}, {1.0/255, 0}, {1.0/255,␣
↪0}};
    engine_image_pre_process.SetConvertAtrr(alpha_beta);
    bool ret = engine_image_pre_process.PushImage(0, 0, BMimg);

    std::map<std::string,sail::Tensor*> output_tensor_map;
    std::vector<sail::BMImage> ost_images;
    int channels = 0;
    std::vector<int> imageidxs;
    std::vector<int> padding_atrr;
    engine_image_pre_process.GetBatchData(output_tensor_map, ost_images, channels,␣
↪imageidxs, padding_atrr);

    std::vector<int> width_list;
    std::vector<int> height_list;
    for (int index = 0; index < channels; index++) {
        width_list.push_back(ost_images[index].width());
        height_list.push_back(ost_images[index].height());
    }

    sail::algo_yolox_post yolox_post(std::vector<std::vector<int>>{{1, 3, 20, 20, 85}, {1, 3,␣
↪40, 40, 85}, {1, 3, 80, 80, 85}}, 640, 640, 10);
    std::vector<float> dete_thresholds = {0.2f, 0.2f, 0.2f};
    std::vector<float> nms_thresholds = {0.5f, 0.5f, 0.5f};
    ret = yolox_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds,␣
↪nms_thresholds, width_list, height_list, padding_atrr);

    std::vector<std::tuple<int, int, int, int, int, float>> detection_results;
    int channel_idx, image_idx;

    std::tie(detection_results, channel_idx, image_idx) = yolox_post.get_result_npy();

    std::cout << "Detection Results:" << std::endl;
    for (const auto& detection : detection_results) {
        int left, top, right, bottom, class_id;
        float score;

        std::tie(left, top, right, bottom, class_id, score) = detection;

        std::cout << "Box: (" << left << ", " << top << ", " << right << ", " << bottom
↪<< ")"
```

(下页继续)

(续上页)

```
            << " Class ID: " << class_id << " Score: " << score << std::endl;
    }

    std::cout << "Channel Index: " << channel_idx << " Image Index: " << image_idx <
↪< std::endl;
    return 0;
}
```

4.36 algo_yolov5_post_cpu_opt

针对 3 输出或 1 输出的 yolov5 模型，对后处理进行了加速。

4.36.1 构造函数

接口形式:

```
algo_yolov5_post_cpu_opt(const std::vector<std::vector<int>>& shape,
                         int network_w=640, int network_h=640);
```

参数说明:

- shape: std::vector<std::vector<int> >

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

4.36.2 process

处理接口。

接口形式 1:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(std::vector
↪<TensorPTRWithName> &input_data,
      std::vector<int> &ost_w,
      std::vector<int> &ost_h,
      std::vector<float> &dete_threshold,
      std::vector<float> &nms_threshold,
      bool input_keep_aspect_ratio,
      bool input_use_multiclass_nms);
```

参数说明 1:

- input_data: std::vector<TensorPTRWithName>

输入参数。输入数据，包含三个输出或一个输出。

- ost_w: std::vector<int>

输入参数。原始图片的宽度。

- ost_h: std::vector<int>

输入参数。原始图片的高度。

- dete_threshold: std::vector<float>

输入参数。检测阈值。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- input_keep_aspect_ratio: bool

输入参数。输入图片是否保持纵横比。

- input_use_multiclass_nms: bool

输入参数。是否用多类 nms。

接口形式 2:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(std::map
↪<std::string, Tensor&>& input_data,
      std::vector<int> &ost_w,
      std::vector<int> &ost_h,
      std::vector<float> &dete_threshold,
      std::vector<float> &nms_threshold,
      bool input_keep_aspect_ratio,
      bool input_use_multiclass_nms);
```

参数说明 2:

- input_data: std::map<std::string, Tensor&>

输入参数。输入数据，包含三个输出或一个输出。

- ost_w: std::vector<int>

输入参数。原始图片的宽度。

- ost_h: std::vector<int>

输入参数。原始图片的高度。

- dete_threshold: std::vector<float>

输入参数。检测阈值。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- input_keep_aspect_ratio: bool

输入参数。输入图片是否保持纵横比。

- input_use_multiclass_nms: bool

输入参数。是否用多类 nms。

返回值说明:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

4.36.3 reset_anchors

更新 anchor 尺寸.

接口形式:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

参数说明:

- anchors_new: std::vector<std::vector<std::vector<int> > >

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```cpp
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
#include <sail/algokit.h>
#include <iostream>
#include <vector>
#include <queue>
#include <numeric>
#include <opencv2/opencv.hpp>

int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg";
    std::string bmodel_name = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/
↪yolov5s_v6.1_3output_int8_1b.bmodel";
    sail::Decoder decoder(image_name, true, tpu_id);
    auto bmimg = decoder.read(handle);
    sail::EngineImagePreProcess engine_image_pre_process(bmodel_name, tpu_id, 0);
    engine_image_pre_process.InitImagePreProcess(sail::sail_resize_type::BM_PADDING_
↪TPU_LINEAR, true, 10, 10);
    engine_image_pre_process.SetPaddingAtrr(114, 114, 114, 1);
    std::vector<std::pair<float, float>> alpha_beta = {{1.0/255, 0}, {1.0/255, 0}, {1.0/255,␣
↪0}};
    engine_image_pre_process.SetConvertAtrr(alpha_beta);
    auto ret = engine_image_pre_process.PushImage(0, 0, bmimg);
    auto output_tensor_map = engine_image_pre_process.GetBatchData(true);
    std::vector<int> width_list;
    std::vector<int> height_list;
    for (int index = 0; index < output_tensor_map.size(); index++) {
        width_list.push_back(output_tensor_map[index].width());
        height_list.push_back(output_tensor_map[index].height());
    }
    auto yolov5_post = sail::algo_yolov5_post_cpu_opt(std::vector<std::vector<int>>{{1,␣
↪3, 20, 20, 85}, {1, 3, 40, 40, 85}, {1, 3, 80, 80, 85}}, 640, 640);
    std::vector<float> dete_thresholds(output_tensor_map.size(), 0.2f);
    std::vector<float> nms_thresholds(output_tensor_map.size(), 0.5f);
    auto objs = yolov5_post.process(output_tensor_map, width_list, height_list, dete_
↪thresholds, nms_thresholds, true, true);
    std::cout << objs << std::endl;
    return 0;
}
```

4.37 tpu_kernel_api_openpose_part_nms

针对 OpenPose 模型，使用智能视觉深度学习处理器 Kernel 对 part nms 后处理进行加速，目前只支持 BM1684x，且 libsophon 的版本必须不低于 0.4.6（v23.03.01）。

4.37.1 构造函数

接口形式:

```
tpu_kernel_api_openpose_part_nms(int device_id,
                                 int network_c,
                                 std::string module_file = "/opt/sophon/libsophon-current/lib/
↪tpu_module/libbm1684x_kernel_module.so");
```

参数说明:

- device_id: int

输入参数。使用的设备编号。

- network_c: int

输入参数。输入通道数，对应关键点通道的数量。

- module_file: string

输入参数。Kernel module 文件路径，默认为”/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

4.37.2 process

处理接口。

接口形式 1:

```
std::tuple<std::vector<std::vector<int>>, std::vector<std::vector<float>>, std::vector
↪<std::vector<int>>> process(
                        TensorPTRWithName& input_data,
                        std::vector<int>& shape,
                        std::vector<float>& threshold,
                        std::vector<int>& max_peak_num);
```

参数说明 1:

- input_data: TensorPTRWithName

输入参数。输入数据。

- shape: std::vector<int>

输入参数。输入数据的宽和高。

- threshold: std::vector<float>

输入参数。检测阈值序列。

- max_peak_num: std::vector<int>

输入参数。最大被检测关键点的数量。

接口形式 2:

```
std::tuple<std::vector<std::vector<int>>, std::vector<std::vector<float>>, std::vector
↪<std::vector<int>>> process(
                        std::map<std::string, Tensor&>& input_data,
                        std::vector<int>& shape,
                        std::vector<float>& threshold,
                        std::vector<int>& max_peak_num);
```

参数说明 2:

- input_data: std::map<std::string, Tensor&>

输入参数。输入数据。

- shape: std::vector<int>

输入参数。输入数据的宽和高。

- threshold: std::vector<float>

输入参数。检测阈值序列。

- max_peak_num: std::vector<int>

输入参数。最大被检测关键点的数量。

返回值说明:

std::tuple<std::vector<std::vector<int🄵>, std::vector<std::vector<float🄵>, std::vector<std::vector<int🄵>>

- 第一个输出: std::vector<std::vector<int🄵>

在每个通道被检测的关键点数量。

- 第二个输出: std::vector<std::vector<float🄵>

所有被检测的关键点的置信度。

- 第三个输出: std::vector<std::vector<int🄵>

所有被检测的关键点的拉平坐标。

4.37.3 reset_network_c

更新关键点通道数。

接口形式:

```
int reset_network_c(int network_c_new);
```

参数说明:

- network_c_new: int

要更新的通道数。

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```
#include <sail/cvwrapper.h>
#include <sail/tpu_kernel_api.h>
#include <opencv2/opencv.hpp>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <math.h>

using namespace std;

int main() {
    int tpu_id = 0;
    std::string image_path = "../../../sophon-demo/sample/OpenPose/datasets/test/3.jpg";
    sail::Decoder decoder(image_path, true, tpu_id);
    std::string bmodel_path = "../../../sophon-demo/sample/OpenPose/models/BM1684/
↪pose_coco_fp32_1b.bmodel";
    sail::Handle handle(tpu_id);
    sail::Engine net(bmodel_path, tpu_id, sail::IOMode::SYSIO);
    cv::Mat src_img = cv::imdecode(std::vector<uchar>(std::ifstream(image_path).
↪readIntoVector()), -1);
    std::string graph_name = net.get_graph_names()[0];
    std::string input_name = net.get_input_names(graph_name)[0];
    std::string output_name = net.get_output_names(graph_name)[0];
    int h, w, _;
    cv::split(src_img, {&h, &w, nullptr});
    int net_h = net.get_input_shape(graph_name, input_name)[2];
    int net_w = net.get_input_shape(graph_name, input_name)[3];
    int out_h = net.get_output_shape(graph_name, output_name)[2];
    float scale = std::min(static_cast<float>(net_h) / h, static_cast<float>(net_w) / w);

    cv::Mat resize_img = cv::resize(src_img, cv::Size(0, 0), scale, scale, cv::INTER_CUBIC);
    cv::Mat pad_img = cv::copyMakeBorder(resize_img, 0, net_h - resize_img.rows, 0, net_
↪w - resize_img.cols, cv::BORDER_CONSTANT, value={114, 114, 114});
```

（下页继续）

(续上页)

```
  std::vector<float> img(net_h * net_w * 3);
  cv::transform(pad_img.data, img.data(), net_h * net_w, 3, [](uchar v) { return (v - 128) /
↪ 255.0f; });

  std::map<std::string, std::vector<float>> outputs = net.process(graph_name, {input_
↪name: img});

  std::vector<float> output(out_h * out_h * 3);
  output = cv2::resize(output, (0, 0), fx=stride, fy=stride, interpolation=cv2::INTER_
↪CUBIC);
  output = output[:resize_img.shape[0], :resize_img.shape[1], :];
  output = cv2::resize(output, (src_img.shape[1], src_img.shape[0]),␣
↪interpolation=cv2::INTER_CUBIC);
  std::vector<float> input_data;
  for (int i = 0; i < output.size(); i += 3) {
      input_data.push_back(cv::GaussianFilter(output.data() + i, sigma=3));
  }
  int point_num = int(net.get_output_shape(graph_name, output_name)[1] / 3) - 1;
  tpu_api_openpose_part_nms_postprocess_t api;

  bm_device_mem_t output_data, output_num;
  assert(BM_SUCCESS == bm_malloc_device_byte(
                       handle, &output_data,
                       sizeof(float) * api.input_c * input_h * input_w));
  assert(BM_SUCCESS == bm_malloc_device_byte(handle,
                                             &output_num,
                                             sizeof(int) * api.input_c));
  api.input_data_addr = bm_mem_get_device_addr(input_data);
  api.output_data_addr = bm_mem_get_device_addr(output_data);
  api.num_output_data_addr = bm_mem_get_device_addr(output_num);

  api.input_h = net_h;
  api.input_w = net_w;
  api.max_peak_num = 96;
  api.nms_thresh = 0.05;

  assert(BM_SUCCESS == tpu_kernel_launch(handle,
                                         func_id, &api, sizeof(api)));
  bm_thread_sync(handle);

  bm_memcpy_d2s_partial(handle, num_result, output_num,
                       sizeof(int) * api.input_c);
  const int peak_num = num_result[api.input_c - 1];
  bm_memcpy_d2s_partial(handle, score_out_result,
                       input_data, peak_num * sizeof(float));
  bm_memcpy_d2s_partial_offset(handle, coor_out_result,
                           input_data, peak_num * sizeof(int),
                           peak_num * sizeof(float));

  bm_free_device(handle, input_data);
  bm_free_device(handle, output_data);
```

(下页继续)

(续上页)

```
    bm_free_device(handle, output_num);
    return 0;
    }
```

4.38 algo_yolov8_post_1output_async

针对以单输出 YOLOv8 模型的后处理接口，内部使用线程池的方式实现。

4.38.1 构造函数

接口形式:

```
algo_yolov8_post_1output_async(const std::vector<int>& shape,
                    int network_w=640,
                    int network_h=640,
                    int max_queue_size=20,
                    bool input_use_multiclass_nms=true,
                    bool agnostic=false);
```

参数说明:

- shape: std::vector<int>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

4.38.2 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

输入参数。输入图像序列的通道号。

- image_idx: std::vector<int>

输入参数。输入图像序列的编号。

- input_data: TensorPTRWithName

输入参数。输入数据。

- dete_threshold: std::vector<float>

输入参数。检测阈值序列。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- ost_w: std::vector<int>

输入参数。原始图片序列的宽。

- ost_h: std::vector<int>

输入参数。原始图片序列的高。

- padding_attrs: std::vector<std::vector<int> >

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

4.38.3 get_result_npy

获取最终的检测结果

接口形式:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
#include <stdio.h>
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
#include <sail/algokit.h>
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <queue>
#include <numeric>

int main() {
    int tpu_id = 0;
```

(下页继续)

(续上页)

```
    sail::Handle handle(tpu_id);
    std::string image_path = "../../../sophon-demo/sample/YOLOv8/datasets/test/3.jpg";
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv8/models/BM1684X/
↪yolov8s_v6.1_1output_int8_4b.bmodel";

    sail::Decoder decoder(image_path, true, tpu_id);
    sail::BMImage bmimg = decoder.read(handle);

    sail::EngineImagePreProcess engine_image_pre_process(bmodel_path, tpu_id, 0);
    engine_image_pre_process.PushImage(0, 0, bmimg);
    std::map<std::string,sail::Tensor*> output_tensor_map;
    std::vector<sail::BMImage> ost_images;
    std::vector<int> channel_list;
    std::vector<int> imageidx_list;
    std::vector<float> padding_atrr;
    engine_image_pre_process.GetBatchData(output_tensor_map, ost_images, channel_list,
↪ imageidx_list, padding_atrr);

    std::queue<std::vector<float>> post_queue;
    std::vector<int> width_list;
    std::vector<int> height_list;
    for (int index = 0; index < channel_list.size(); index++) {
        width_list.push_back(ost_images[index].width());
        height_list.push_back(ost_images[index].height());
    }
    post_queue.push(std::vector<float>({output_tensor_map, channel_list, imageidx_list,␣
↪width_list, height_list, padding_atrr}));

    sail::algo_yolov8_post_1output_async yolov8_post([4, 84, 8400], 640, 640, 10);
    std::vector<float> dete_thresholds(channels.size(), 0.2);
    std::vector<float> nms_thresholds(channels.size(), 0.5);
    yolov8_post.push_data(channel_list, imageidx_list, output_tensor_map, dete_
↪thresholds, nms_thresholds, width_list, height_list, padding_atrr);
    std::vector<std::tuple<int, int, int, int ,int, float>> objs;
    std::vector<int> channel;
    std::vector<int> image_idx;
    yolov8_post.get_result(&objs, &channel, &image_idx);
    std::cout << "objs: " << objs << ", channel: " << channel << ", image idx: " <<␣
↪image_idx << std::endl;

    return 0;
}
```

4.39 algo_yolov8_post_cpu_opt_1output_async

针对以单输出 YOLOv8 模型的后处理接口，内部使用线程池的方式实现。

4.39.1 构造函数

接口形式:

```
algo_yolov8_post_cpu_opt_1output_async(const std::vector<int>& shape,
                    int network_w=640,
                    int network_h=640,
                    int max_queue_size=20,
                    bool input_use_multiclass_nms=true,
                    bool agnostic=false);
```

参数说明:

- shape: std::vector<int>

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

4.39.2 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
```

(下页继续)

(续上页)

```
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

输入参数。输入图像序列的通道号。

- image_idx: std::vector<int>

输入参数。输入图像序列的编号。

- input_data: TensorPTRWithName

输入参数。输入数据。

- dete_threshold: std::vector<float>

输入参数。检测阈值序列。

- nms_threshold: std::vector<float>

输入参数。nms 阈值序列。

- ost_w: std::vector<int>

输入参数。原始图片序列的宽。

- ost_h: std::vector<int>

输入参数。原始图片序列的高。

- padding_attrs: std::vector<std::vector<int> >

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

4.39.3 get_result_npy

获取最终的检测结果

接口形式:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
#include <stdio.h>
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
#include <sail/algokit.h>
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <queue>
#include <numeric>

int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_path = "../../../sophon-demo/sample/YOLOv8/datasets/test/3.jpg";
    std::string bmodel_path = "../../../sophon-demo/sample/YOLOv8/models/BM1684X/
↪yolov8s_v6.1_1output_int8_4b.bmodel";

    sail::Decoder decoder(image_path, true, tpu_id);
    sail::BMImage bmimg = decoder.read(handle);

    sail::EngineImagePreProcess engine_image_pre_process(bmodel_path, tpu_id, 0);
    engine_image_pre_process.PushImage(0, 0, bmimg);
    std::map<std::string,sail::Tensor*> output_tensor_map;
    std::vector<sail::BMImage> ost_images;
```

(下页继续)

(续上页)

```
    std::vector<int> channel_list;
    std::vector<int> imageidx_list;
    std::vector<float> padding_atrr;
    engine_image_pre_process.GetBatchData(output_tensor_map, ost_images, channel_list,
↪ imageidx_list, padding_atrr);

    std::queue<std::vector<float>> post_queue;
    std::vector<int> width_list;
    std::vector<int> height_list;
    for (int index = 0; index < channel_list.size(); index++) {
        width_list.push_back(ost_images[index].width());
        height_list.push_back(ost_images[index].height());
    }
    post_queue.push(std::vector<float>({output_tensor_map, channel_list, imageidx_list,
↪width_list, height_list, padding_atrr}));

    sail::algo_yolov8_post_cpu_opt_1output_async yolov8_post([4, 84, 8400], 640, 640, 10);
    std::vector<float> dete_thresholds(channels.size(), 0.2);
    std::vector<float> nms_thresholds(channels.size(), 0.5);
    yolov8_post.push_data(channel_list, imageidx_list, output_tensor_map, dete_
↪thresholds, nms_thresholds, width_list, height_list, padding_atrr);
    std::vector<std::tuple<int, int, int, int ,int, float>> objs;
    std::vector<int> channel;
    std::vector<int> image_idx;
    yolov8_post.get_result(&objs, &channel, &image_idx);
    std::cout << "objs: " << objs << ", channel: " << channel << ", image idx: " <<
↪image_idx << std::endl;

    return 0;
}
```

CHAPTER 5

SAIL Python API 参考

5.1 Basic function

主要用于获取或配置设备信息与属性。

5.1.1 get_available_tpu_num

获取当前设备中可用智能视觉深度学习处理器的数量。

接口形式:

```
def get_available_tpu_num() -> int
```

返回值说明:

返回当前设备中可用智能视觉深度学习处理器的数量。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    tpu_len = sail.get_available_tpu_num()
    print('available tpu:',tpu_len)
```

5.1.2 set_print_flag

设置是否打印程序的计算耗时信息。

接口形式:

```
def set_print_flag(print_flag: bool) -> None
```

参数说明:

- print_flag: bool

print_flag 为 True 时，打印程序的计算主要的耗时信息，否则不打印。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    set_print_flag(True)
```

5.1.3 set_dump_io_flag

设置是否存储输入数据和输出数据。

接口形式:

```
def set_dump_io_flag(dump_io_flag: bool) -> None
```

参数说明:

- dump_io_flag: bool

dump_io_flag 为 True 时，存储输入数据和输出数据，否则不存储。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    set_dump_io_flag(True)
```

5.1.4 set_loglevel

设置运行过程中的日志级别为指定级别。较低的日志等级通常用于生产环境，以减少性能开销和日志数据量，而较高的日志等级则适用于开发和调试，以便能够记录更详细的信息。

接口形式:

```
def set_loglevel(sail.LogLevel loglevel) -> int
```

参数说明:

- loglevel: LogLevel

期望的日志级别，为 sail.LogLevel 枚举值。可选的级别包括 TRACE、DEBUG、INFO、WARN、ERROR、CRITICAL、OFF，默认级别为 INFO。

返回值说明:

返回类型：int

0：日志级别设置成功。-1：传入了未知的日志级别，设置失败。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    ret = sail.set_loglevel(sail.LogLevel.TRACE)
    if (ret == 0):
        print("Set log level successfully")
    else:
        print("Unknown log level, set failed.")
```

5.1.5 set_decoder_env

设置 Decoder 的环境变量，必须在 Decoder 构造前设置，否则使用默认值。

接口形式:

```
def set_decoder_env(env_name: str, env_value: str) -> None
```

参数说明:

- env_name: str

选择设置 Decoder 的属性名称，可选的属性名称有:

- ‘refcounted_frames’ 设置为 1 时，解码出来的图像需要程序手动释放，为 0 时由 Decoder 自动释放。
- ‘extra_frame_buffer_num’ 设置 Decoder 的最大缓存帧数
- ‘rtsp_transport’ 设置 RTSP 采用的传输协议
- ‘stimeout’ 设置阻塞超时时间
- ‘rtsp_flags’ 设置 RTSP 是否自定义 IO
- ‘buffer_size’ 设置缓存大小
- ‘max_delay’ 设置最大时延
- ‘probesize’ 解析文件时读取的最大字节数
- ‘analyzeduration’ 解析文件时读取的最大时长

- env_value: str

该属性的配置值

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    sail.set_decoder_env("extra_frame_buffer_num", "1")
    dev_id = 0
    handle = sail.Handle(dev_id)
    image_name = "your_image.jpg"
    decoder = sail.Decoder(image_name, False, dev_id)
    bmimg1 = sail.BMImage(handle, 256, 256, sail.Format.FORMAT_BGR_PLANAR, sail.
↪ImgDtype.DATA_TYPE_EXT_1N_BYTE)
    decoder.read(handle,bmimg1)
```

5.1.6 base64_encode

将字节数据进行 base64 编码，返回 bytes 类型的编码数据

接口形式:

```
def base64_encode(handle: Handle, input_bytes: bytes) -> bytes:
```

参数说明:

- handle: sail.Handle

设备的 handle 句柄，使用 sail.Handle(dev_id) 创建

- input_bytes: bytes

待编码的字节数据

返回值说明

返回 base64 编码的字节数据

示例代码

```
import sophon.sail as sail
import numpy as np

if __name__ == "__main__":
    # 示例 NumPy 数组
    arr = np.array([[1, 2, 3], [4, 5, 6]])
    # 转为字节数据
    arr_bytes = arr.tobytes()
    # 创建handle，soc设备默认为dev_id为0
    handle = sail.Handle(0)
    # base字节编码
    base64_encoded_arr = sail.base64_encode(handle,arr_bytes)
```

5.1.7 base64_decode

将 base64 的字节编码数据进行解码，返回解码后的字节数据

接口形式：

```
def base64_decode(handle: Handle, encode_bytes: bytes) -> bytes:
```

参数说明：

- handle: sail.Handle

设备的 handle 句柄，使用 sail.Handle(dev_id) 创建

- encode_bytes: bytes

base64 的字节编码数据

返回值说明

返回 base64 解码的字节数据

示例代码

```
import sophon.sail as sail
import numpy as np

if __name__ == "__main__":
    # 示例 NumPy 数组
    arr = np.array([[1, 2, 3], [4, 5, 6]])
    shape = arr.shape
    # 转为字节数据
    arr_bytes = arr.tobytes()
    # 创建handle，soc设备默认为dev_id为0
    handle = sail.Handle(0)
    # base字节编码
    base64_encoded_arr = sail.base64_encode(handle,arr_bytes)

    # 解码数据
    base64_decode_arr = sail.base64_decode(handle,base64_encoded_arr)
    # 将生成byte数据转换为numpy数据
    res_arr = np.frombuffer(arr_bytes, dtype=np.int64).reshape(shape)
```

5.1.8 base64_encode_array

对 numpy.array 进行 base64 编码，生成字节编码数据。

示例代码请参考 **base64_decode_asarray** 接口提供的示例代码

接口形式：

```
def base64_encode_array(handle: Handle, input_arr: numpy.ndarray) -> bytes:
```

参数说明：

- handle: sail.Handle

设备的 handle 句柄，使用 sail.Handle(dev_id) 创建

- input_arr: numpy.ndarray

待编码的 numpy.ndarray 数据

返回值说明

返回 base64 解码的字节数据

5.1.9 base64_decode_asarray

base64 解码，生成 numpy.array 数据

接口形式：

```
def base64_decode_asarray(handle: Handle, encode_arr_bytes: bytes, array_type:str =
↪"uint8") -> numpy.ndarray:
```

参数说明：

- handle: sail.Handle

设备的 handle 句柄，使用 sail.Handle(dev_id) 创建

- encode_arr_bytes: bytes

base64 编码后的 numpy.ndarray 的字节数据

- dtype: str

numpy.ndarray 的数据类型，默认 uint8，支持 float、uint8、int8、int16、int32、int64

返回值说明

返回 base64 解码的一维 numpay.array 数组

示例代码

```
import sophon.sail as sail
import numpy as np

if __name__ == "__main__":
    # 示例 NumPy 数组
    arr = np.array([[1,2,3],[4,5,6]],dtype=np.uint8)
    # base64编码
    base64_encoded = sail.base64_encode_array(handle,arr)
    # base64解码
    res_array = sail.base64_decode_asarray(handle,base64_encoded).reshape(shape)
```

5.1.10 get_tpu_util

获取对应设备的处理器使用率

接口形式:

```
def get_tpu_util(dev_id: int) -> int
```

参数说明:

- dev_id: int

需要获取处理器使用率的设备的 ID。

返回值说明:

返回对应设备的处理器使用率百分比。

示例代码

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("dev {} tpu-util is {} %".format(dev_id,sail.get_tpu_util(dev_id)))
```

5.1.11 get_vpu_util

获取对应设备的 VPU 使用率

接口形式:

```
def get_vpu_util(dev_id: int) -> list
```

参数说明:

- dev_id: int

需要获取 VPU 使用率的设备的 ID。

返回值说明:

bm1684 为 5 核 vpu，返回值为长度为 5 的 List，bm1684x 为 3 核 vpu，返回值为长度为 3 的 List。List 中的每项数据为对应核心的使用率百分比。

示例代码

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_vpu_util",sail.get_vpu_util(dev_id))
```

5.1.12 get_vpp_util

获取对应设备的 VPP 使用率

接口形式:

```
def get_vpp_util(dev_id: int) -> list
```

参数说明:

- dev_id: int

需要获取 VPP 使用率的设备的 ID。

返回值说明:

bm1684 与 bm1684x 均为 2 核 vpp，返回值为长度为 2 的 List。List 中的每项数据为对应核心的使用率百分比。

示例代码

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_vpp_util",sail.get_vpp_util(dev_id))
```

5.1.13 get_board_temp

接口形式:

```
def get_board_temp(dev_id: int) -> int
```

参数说明:

- dev_id: int

需要获取对应板卡所在设备的 ID。

返回值说明:

返回对应板卡的板级温度，默认单位摄氏度（℃）。

示例代码

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_board_temp",sail.get_board_temp(dev_id))
```

5.1.14 get_chip_temp

接口形式:

```
def get_chip_temp(dev_id: int) -> int
```

参数说明:

- dev_id: int

需要获取处理器温度的设备的 ID。

返回值说明:

返回对应设备的处理器的温度。，默认单位摄氏度（℃）。

示例代码

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_chip_temp",sail.get_chip_temp(dev_id))
```

5.1.15 get_dev_stat

接口形式:

```
def get_dev_stat(dev_id: int) -> list
```

参数说明:

- dev_id: int

需要获取内存信息的设备的 ID。

返回值说明:

返回对应设备的内存信息列表:[mem_total,mem_used,tpu_util]。

示例代码

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_dev_stat",sail.get_dev_stat(dev_id))
```

5.2 sail.Data type

定义 SOPHON 环境中常用的数据类型

接口形式:

```
sail.Dtype.BM_FLOAT32
sail.Dtype.BM_INT8
sail.Dtype.BM_UINT8
sail.Dtype.BM_INT32
sail.Dtype.BM_UINT32
sail.Dtype.BM_FLOAT16,
sail.Dtype.BM_BFLOAT16,
sail.Dtype.BM_INT16,
sail.Dtype.BM_UINT16
```

参数说明:

- sail.Dtype.BM_FLOAT32 数据类型为 float32
- sail.Dtype.BM_INT8 数据类型为 int8
- sail.Dtype.BM_UINT8 数据类型为 uint8
- sail.Dtype.BM_INT32 数据类型为 int32
- sail.Dtype.BM_UINT32 数据类型为 uint32
- BM_FLOAT16 数据类型为 uint32
- BM_BFLOAT16 数据类型为 uint32
- BM_INT16 数据类型为 uint32
- BM_UINT16 数据类型为 uint32

5.3 sail.PaddingAtrr

PaddingAtrr 中存储了数据 padding 的各项属性，可通过配置 PaddingAtrr 进行数据填充

5.3.1 __init__

初始化 PaddingAtrr

接口形式:

```
def __init__(self)

def __init__(self, stx: int, sty: int, width: int, height: int, r: int, g: int, b: int)
```

参数说明:

- stx: int

原图像相对于目标图像在 x 方向上的偏移量

- sty: int

原图像相对于目标图像在 y 方向上的偏移量

- width: int

在 padding 的同时可对原图像进行 resize，width 为原图像 resize 后的宽，若不进行 resize，则 width 为原图像的宽

- height: int

在 padding 的同时可对原图像进行 resize，height 为原图像 resize 后的高，若不进行 resize，则 height 为原图像的高

- r: int

padding 时在 R 通道上填充的像素值

- g: int

padding 时在 G 通道上填充的像素值

- b: int

padding 时在 B 通道上填充的像素值

5.3.2 set_stx

设置原图像相对于目标图像在 x 方向上的偏移量

接口形式:

```
def set_stx(self, stx: int) -> None
```

参数说明:

- stx: int

原图像相对于目标图像在 x 方向上的偏移量

5.3.3 set_sty

设置原图像相对于目标图像在 y 方向上的偏移量

接口形式:

```
def set_sty(self, sty: int) -> None
```

参数说明:

- sty: int

原图像相对于目标图像在 y 方向上的偏移量

5.3.4 set_w

设置原图像 resize 后的 width

接口形式:

```
def set_w(self, width: int) -> None
```

参数说明:

- width: int

在 padding 的同时可对原图像进行 resize，width 为原图像 resize 后的宽，若不进行 resize，则 width 为原图像的宽

5.3.5 set_h

设置原图像 resize 后的 height

接口形式:

```
def set_h(self, height: int) -> None
```

参数说明:

- height: int

在 padding 的同时可对原图像进行 resize，height 为原图像 resize 后的高，若不进行 resize，则 height 为原图像的高

5.3.6 set_r

设置 R 通道上的 padding 值

接口形式:

```
def set_r(self, r: int) -> None
```

参数说明

- r: int

R 通道上的 padding 值

5.3.7 set_g

设置 G 通道上的 padding 值

接口形式:

```
def set_g(self, g: int) -> None
```

参数说明:

- g: int

G 通道上的 padding 值

5.3.8 set_b

设置 B 通道上的 padding 值

接口形式:

```
def set_b(self, b: int) -> None
```

参数说明

- b: int

B 通道上的 padding 值

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    bmcv = sail.Bmcv(handle)
    paddingatt = sail.PaddingAtrr()
    paddingatt.set_stx(0)
    paddingatt.set_sty(0)
    paddingatt.set_w(resize_w)
    paddingatt.set_h(resize_h)
    paddingatt.set_r(114)
    paddingatt.set_g(114)
    paddingatt.set_b(114)
    output_temp = bmcv.crop_and_resize_padding(input,0,0,image_w,image_h,resize_w,
↪resize_h,paddingatt)
```

5.4 sail.Handle

Handle 是设备句柄的包装类，在程序中用于设备的标识。

5.4.1 __init__

初始化 Handle

接口形式:

```
def __init__(self, tpu_id: int)
```

参数说明:

- tpu_id: int

创建 Handle 使用的智能视觉深度学习处理器的 id 号

5.4.2 get_device_id

获取 Handle 中智能视觉深度学习处理器的 id

接口形式:

```
def get_device_id(self) -> int
```

返回值说明:

- tpu_id: int

Handle 中的智能视觉深度学习处理器的 id 号

5.4.3 get_sn

获取 Handle 中标识设备的序列码

接口形式:

```
def get_sn(self) -> str
```

返回值说明:

- serial_number: str

返回 Handle 中设备的序列码

5.4.4 get_target

获取设备的智能视觉深度学习处理器型号

接口形式:

```
def get_target(self) -> str
```

返回值说明:

- Tensor Computing Processor type: str

返回设备智能视觉深度学习处理器的型号

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    print(handle.get_device_id(), handle.get_sn(), handle.get_target())
```

5.5 sail.IOMode

IOMode 用于定义输入 Tensor 和输出 Tensor 的内存位置信息（device memory 或 system memory）。

接口形式:

```
sail.IOMode.SYSI
sail.IOMode.SYSO
sail.IOMode.SYSIO
sail.IOMode.DEVIO
```

参数说明:

- sail.IOMode.SYSI

输入 Tensor 在 system memory，输出 Tensor 在 device memory

- sail.IOMode.SYSO

输入 Tensor 在 device memory，输出 Tensor 在 system memory

- sail.IOMode.SYSIO

输入 Tensor 在 system memory，输出 Tensor 在 system memory

- sail.IOMode.DEVIO

输入 Tensor 在 device memory，输出 Tensor 在 device memory

5.6 sail.LogLevel

LogLevel 用于定义日志级别。最高为 TRACE ， 最低为 OFF ， 默认为 INFO 。

接口形式:

```
sail.LogLevel.TRACE
sail.LogLevel.DEBUG
sail.LogLevel.INFO
sail.LogLevel.WARN
sail.LogLevel.ERROR
sail.LogLevel.CRITICAL
sail.LogLevel.OFF
```

参数说明:

- TRACE

打印 TRACE 级别和更低级别的日志。

- DEBUG

打印 DEBUG 级别和更低级别的日志。

- INFO

打印 INFO 级别和更低级别的日志。

- WARN

打印 WARN 级别和更低级别的日志。

- ERROR

打印 ERROR 级别和更低级别的日志。

- CRITICAL

打印 CRITICAL 级别和更低级别的日志。

- OFF

关闭各个级别的日志打印。

5.7 sail.bmcv_resize_algorithm

定义图像 resize 中常见的插值策略

接口形式:

```
sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST
sail.bmcv_resize_algorithm.BMCV_INTER_LINEAR
sail.bmcv_resize_algorithm.BMCV_INTER_BICUBIC
```

参数说明

· sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

最近邻插值算法

· sail.bmcv_resize_algorithm.BMCV_INTER_LINEAR

双线性插值算法

· sail.bmcv_resize_algorithm.BMCV_INTER_BICUBIC

双三次插值算法

5.8 sail.Format

定义常用的图像格式。

接口形式:

```
sail.Format.FORMAT_YUV420P
sail.Format.FORMAT_YUV422P
sail.Format.FORMAT_YUV444P
sail.Format.FORMAT_NV12
sail.Format.FORMAT_NV21
sail.Format.FORMAT_NV16
sail.Format.FORMAT_NV61
sail.Format.FORMAT_NV24
sail.Format.FORMAT_RGB_PLANAR
sail.Format.FORMAT_BGR_PLANAR
sail.Format.FORMAT_RGB_PACKED
sail.Format.FORMAT_BGR_PACKED
sail.Format.FORMAT_RGBP_SEPARATE
sail.Format.FORMAT_BGRP_SEPARATE
sail.Format.FORMAT_GRAY
sail.Format.FORMAT_COMPRESSED
```

参数说明:

· sail.Format.FORMAT_YUV420P

表示预创建一个 YUV420 格式的图片，有三个 plane

· sail.Format.FORMAT_YUV422P

表示预创建一个 YUV422 格式的图片，有三个 plane

· sail.Format.FORMAT_YUV444P

表示预创建一个 YUV444 格式的图片，有三个 plane

· sail.Format.FORMAT_NV12

表示预创建一个 NV12 格式的图片，有两个 plane

· sail.Format.FORMAT_NV21

表示预创建一个 NV21 格式的图片，有两个 plane

- sail.Format.FORMAT_NV16

表示预创建一个 NV16 格式的图片，有两个 plane

- sail.Format.FORMAT_NV61

表示预创建一个 NV61 格式的图片，有两个 plane

- sail.Format.FORMAT_RGB_PLANAR

表示预创建一个 RGB 格式的图片，RGB 分开排列, 有一个 plane

- sail.Format.FORMAT_BGR_PLANAR

表示预创建一个 BGR 格式的图片，BGR 分开排列, 有一个 plane

- sail.Format.FORMAT_RGB_PACKED

表示预创建一个 RGB 格式的图片，RGB 交错排列, 有一个 plane

- sail.Format.FORMAT_BGR_PACKED

表示预创建一个 BGR 格式的图片，BGR 交错排列, 有一个 plane

- sail.Format.FORMAT_RGBP_SEPARATE

表示预创建一个 RGB planar 格式的图片，RGB 分开排列并各占一个 plane，共有 3 个 plane

- sail.Format.FORMAT_BGRP_SEPARATE

表示预创建一个 BGR planar 格式的图片，BGR 分开排列并各占一个 plane，共有 3 个 plane

- sail.Format.FORMAT_GRAY

表示预创建一个灰度图格式的图片，有一个 plane

- sail.Format.FORMAT_COMPRESSED

表示预创建一个 VPU 内部压缩格式的图片，共有四个 plane，分别存放内容如下：

plane0: Y 压缩表

plane1: Y 压缩数据

plane2: CbCr 压缩表

plane3: CbCr 压缩数据

5.9 sail.ImgDtype

定义几种图像的存储形式。

接口形式:

```
sail.ImgDtype.DATA_TYPE_EXT_FLOAT32
sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE
sail.ImgDtype.DATA_TYPE_EXT_4N_BYTE
sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE_SIGNED
sail.ImgDtype.DATA_TYPE_EXT_4N_BYTE_SIGNED
```

参数说明:

- sail.ImgDtype.DATA_TYPE_EXT_FLOAT32

表示图片的数据类型为 float32。

- sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE

表示图片的数据类型为 uint8。

- sail.ImgDtype.DATA_TYPE_EXT_4N_BYTE

表示图片的数据类型为 uint8，且每 4 张图片的数据交错排列，数据读写效率更高。

- sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE_SIGNED

表示图片的数据类型为 int8。

- sail.ImgDtype.DATA_TYPE_EXT_4N_BYTE_SIGNED

表示图片的数据类型为 int8，且每 4 张图片的数据交错排列，数据读写效率更高。

5.10 sail.Tensor

Tensor 是模型推理的输入输出类型，包含了数据信息，实现内存管理。

5.10.1 __init__

初始化 Tensor, 并为 Tensor 分配内存

接口形式 1:

```
def __init__(self, handle: Handle, data: np.array, own_sys_data=True)
```

参数说明 1:

- handle: sail.Handle

设备标识 Handle

- array_data: numpy.array

利用 numpy.array 类型初始化 Tensor，其数据类型可以是 np.float32,np.int8,np.uint8

- own_sys_data: bool

指示该 Tensor 是否拥有 system memory，如果为 False，则直接将数据复制到 device memory

接口形式 2

```
def __init__(self, handle: Handle, shape: list[int], dtype: Dtype, own_sys_data: bool, own_
↪dev_data: bool)
```

参数说明 2:

- handle: sail.Handle

设备标识 Handle

- shape: tuple

设置 Tensor 的 shape

- dtype: sail.Dtype

Tensor 的数据类型

- own_sys_data: bool

指示 Tensor 是否拥有 system memory

- own_dev_data: bool

指示 Tensor 是否拥有 device memory

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor2 = sail.Tensor(handle,[1,2],sail.Dtype.BM_FLOAT32,true,true)
```

5.10.2 shape

获取 Tensor 的 shape

接口形式:

```
def shape(self) -> list :
```

返回值说明:

- tensor_shape : list

返回 Tensor 的 shape 的列表。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.shape())
```

5.10.3 dtype

获取 Tensor 的数据类型

接口形式:

```
def dtype(self) -> sail.Dtype :
```

返回值说明:

- data_type : sail.Dtype

返回 Tensor 的数据类型。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.dtype())
```

5.10.4 asnumpy

获取 Tensor 中系统内存的数据，返回 numpy.array 类型。

接口形式:

```
def asnumpy(self) -> numpy.array

def asnumpy(self, shape: tuple) -> numpy.array
```

参数说明:

- shape: tuple

可对 Tensor 中的数据 reshape，返回形状为 shape 的 numpy.array

返回值说明

返回 Tensor 中系统内存的数据，返回类型为 numpy.array。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_ = input_tensor1.asnumpy()
    input__ = input_tensor1.asnumpy((3,1))
```

5.10.5 update_data

更新 Tensor 中系统内存的数据，如果没有分配系统内存，则更新设备内存中的数据。

接口形式:

```
def update_data(self, data: numpy.array) -> None
```

参数说明:

- data: numpy.array

更新的数据，数据类型应和 Tensor 一致，数据 size 不能超过 Tensor 的 size，Tensor 的 shape 将保持不变。

注：如果是 numpy.float16 类型的数据，应使用 numpy.view(numpy.uint16) 再传递给本接口。

示例代码:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)

    tensor_fp32 = sail.Tensor(handle, [1,3,640,640], sail.BM_FLOAT32)
    np_fp32 = np.ones(tensor_fp32.shape(),dtype=np.float32)
    tensor_fp32.update_data(np_fp32)

    tensor_fp16 = sail.Tensor(handle, [1,3,640,640], sail.BM_FLOAT16)
    np_fp16 = np.ones(tensor_fp16.shape(),dtype=np.float16)
    tensor_fp16.update_data(np_fp16.view(np.uint16))
```

5.10.6 scale_from

先对 data 按比例缩放，再将数据更新到 Tensor 的系统内存。

接口形式:

```
def scale_from(self, data: numpy.array, scale: float32)->None
```

参数说明:

- data: numpy.array

对 data 进行 scale，再将数据更新到 Tensor 的系统内存。

- scale: float32

等比例缩放时的尺度。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor1_ = input_tensor1.scale_from(input,0.1)
```

5.10.7 scale_to

先对 Tensor 进行等比例缩放，再将数据返回到系统内存。

接口形式:

```
def scale_to(self, scale: float32)->numpy.array

def scale_to(self, scale: float32, shape: tuple)->numpy.array
```

参数说明:

- scale: float32

等比例缩放时的尺度。

- shape: tuple

数据返回前可进行 reshape，返回 shape 形状的数据。

返回值说明:

- data: numpy.array

将处理后的数据返回至系统内存，返回 numpy.array

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor1_ = input_tensor1.scale_to(input,0.1)
    input_tensor1__ = input_tensor1.scale_to(input,0.1,(3,1))
```

5.10.8 reshape

对 Tensor 进行 reshape

接口形式:

```
def reshape(self, shape: list)->None
```

参数说明:

- shape: list

设置期望得到的新 shape。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor1_ = input_tensor1.reshape([3,1])
```

5.10.9 own_sys_data

查询该 Tensor 是否拥有系统内存的数据指针。

接口形式:

```
def own_sys_data(self)->bool
```

返回值说明:

- judge_ret: bool

如果拥有系统内存的数据指针则返回 True，否则 False。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.own_sys_data())
```

5.10.10 own_dev_data

查询该 Tensor 是否拥有设备内存的数据

接口形式:

```
def own_dev_data(self)->bool
```

返回值说明:

- judge_ret : bool

如果拥有设备内存中的数据则返回 True，否则 False。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.own_dev_data())
```

5.10.11 sync_s2d

将 Tensor 中的数据从系统内存拷贝到设备内存。

接口形式:

```
def sync_s2d(self)->None

def sync_s2d(self, size)->None
```

参数说明:

- size: int

将特定 size 字节的数据从系统内存拷贝到设备内存。

接口形式:

```
def sync_s2d(self, src: sail.Tensor, offset_src: int, offset_dst: int, len: int)->None
```

参数说明:

- src: sail.Tensor

指定被拷贝的 Tensor。

- offset_src: int

指定被拷贝 Tensor 上的数据偏移几个元素后开始拷贝。

- offset_dst: int

指定拷贝目标 Tensor 上的数据偏移几个元素后开始拷贝。

- len: int

指定拷贝长度，既拷贝的元素个数。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor2 = sail.Tensor(handle,[1,2],sail.Dtype.BM_FLOAT32,true,true)
    input_tensor2.sync_s2d()
    input_tensor2.sync_s2d(1)
    input_tensor2.sync_s2d(input_tensor1,0,0,2)
```

5.10.12 sync_d2s

将 Tensor 中的数据从设备内存拷贝到系统内存。

接口形式:

```
def sync_d2s(self)->None

def sync_d2s(self, size: int)->None
```

参数说明:

- size: int

将特定 size 字节的数据从设备内存拷贝到系统内存。

接口形式:

```
def sync_d2s(self, src: sail.Tensor, offset_src: int, offset_dst: int, len: int)->None
```

参数说明:

· src: sail.Tensor

指定被拷贝的 Tensor。

· offset_src: int

指定被拷贝 Tensor 上的数据偏移几个元素后开始拷贝。

· offset_dst: int

指定拷贝目标 Tensor 上的数据偏移几个元素后开始拷贝。

· len: int

指定拷贝长度，既拷贝的元素个数。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)

    input_tensor1 = sail.Tensor(handle,[1,3],sail.Dtype.BM_FLOAT32,False,True)
    input_tensor2 = sail.Tensor(handle,[1,3],sail.Dtype.BM_FLOAT32,True,True)

    input_tensor1.ones()
    input_tensor2.sync_d2s()
    input_tensor2.sync_d2s(2)
    input_tensor2.sync_d2s(input_tensor1,0,0,2)
```

5.10.13 sync_d2d

将另外一个 Tensor 设备内存上的数据拷贝到本 Tensor 的设备内存中。

接口形式:

```
def sync_d2d(self, src: sail.Tensor, offset_src: int, offset_dst: int, len: int)->None
```

参数说明:

· src: sail.Tensor

指定被拷贝的 Tensor。

· offset_src: int

指定被拷贝 Tensor 上的数据偏移几个元素后开始拷贝。

· offset_dst: int

指定拷贝目标 Tensor 上的数据偏移几个元素后开始拷贝。

· len: int

指定拷贝长度，既拷贝的元素个数。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    handle_ = sail.Handle(1)
    input_tensor1 = sail.Tensor(handle,[1,3],sail.Dtype.BM_FLOAT32,False,True)
    input_tensor2 = sail.Tensor(handle_,[1,3],sail.Dtype.BM_FLOAT32,True,True)

    input_tensor1.ones()
    input_tensor2.sync_d2d(input_tensor1,0,0,2)
```

5.10.14 dump_data

将 Tensor 中的数据写入到指定文件中

接口形式:

```
def dump_data(file_name: str, bin: bool = False)
```

参数说明:

- file_name: str

写入文件的路径

- bin: bool

是否采用二进制的形式存储 Tensor, 默认 false.

示例代码:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    data = np.ones([1,20],dtype=int)
    ts = sail.Tensor(handle,data)
    ts.scale_from(data,[0.01,0.1])
    ts.dump_data("./temp.txt")
    ret_data = np.loadtxt("./temp.txt")
    print(ts.asnumpy(),ret_data)
```

5.10.15 memory_set

将本 Tensor 的数据全部置为 c，在接口内部根据本 Tensor 的 dtype 对 c 做相应的类型转换。

接口形式:

```
def memory_set(self, c: any)->None
```

参数说明:

- c: any

需要填充的值。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = 1
    input_tensor1 = sail.Tensor(handle,[1],sail.Dtype.BM_FLOAT32,True,True)

    input_tensor1.memory_set(input)
```

5.10.16 zeros

将本 Tensor 的数据全部置为 0。

接口形式:

```
def zeros(self)->None
```

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

    input_tensor1.zeros()
```

5.10.17 ones

将本 Tensor 的数据全部置为 1。

接口形式:

```
def ones(self)->None
```

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

    input_tensor1.ones()
```

5.11 sail.Engine

Engine 可以实现 bmodel 的加载与管理，是实现模型推理的主要模块。

5.11.1 __init__

初始化 Engine

接口形式 1:

创建 Engine 实例，并不加载 bmodel

```
def __init__(tpu_id: int)

def __init__(self, handle: sail.Handle)
```

参数说明 1:

- tpu_id: int

指定 Engine 实例使用的智能视觉深度学习处理器的 id

- handle: sail.Handle

指定 Engine 实例使用的设备标识 Handle

接口形式 2:

创建 Engine 实例并加载 bmodel，需指定 bmodel 路径或内存中的位置。

```
def __init__(self, bmodel_path: str, tpu_id: int, mode: sail.IOMode)

def __init__(self, bmodel_bytes: bytes, bmodel_size: int, tpu_id: int, mode: sail.
↪IOMode)
```

(下页继续)

(续上页)

参数说明 2:

- bmodel_path: str

指定 bmodel 文件的路径

- tpu_id: int

指定 Engine 实例使用的智能视觉深度学习处理器的 id

- mode: sail.IOMode

指定输入/输出 Tensor 所在的内存位置：系统内存或设备内存。

- bmodel_bytes: bytes

bmodel 在系统内存中的 bytes。

- bmodel_size: int

bmodel 在内存中的字节数

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine1 = sail.Engine(0)

    handle = sail.Handle(0)
    engine2 = sail.Engine(bmodel_path,handle)

    engine3 = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)

    file = open(bmodel_path,"rb")
    datas = file.read()
    file_size = os.path.getsize(bmodel_path)
    engine4 = sail.Engine(datas,file_size,0,sail.IOMode.SYSI)
```

5.11.2 get_handle

获取 Engine 中使用的设备句柄 sail.Handle

接口形式:

```
def get_handle(self)->sail.Handle
```

返回值说明:

- handle: sail.Handle

返回 Engine 中的设备句柄。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    engine1 = sail.Engine(0)
    handle = engine1.get_handle()
```

5.11.3 load

将 bmodel 载入 Engine 中。

接口形式 1:

指定 bmodel 路径，从文件中载入 bmodel。

```
def load(self, bmodel_path: str)->bool
```

参数说明 1:

- bmodel_path: str

bmodel 的文件路径

接口形式 2:

从系统内存中载入 bmodel。

```
def load(self, bmodel_bytes: bytes, bmodel_size: int)->bool
```

参数说明 2:

- bmodel_bytes: bytes

bmodel 在系统内存中的 bytes。

- bmodel_size: int

bmodel 在内存中的字节数。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine1 = sail.Engine(0)
    engine1.load(bmodel_path)
```

5.11.4 get_graph_names

获取 Engine 中所有载入的计算图的名称。

接口形式:

```
def get_graph_names(self)->list
```

返回值说明:

- graph_names: list

Engine 中所有计算图的 name 的列表。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine1 = sail.Engine(0)
    engine1.load(bmodel_path)
    graph_names = engine1.get_graph_names()
```

5.11.5 set_io_mode

设置 Engine 的输入/输出 Tensor 所在的内存位置：系统内存或设备内存。

接口形式:

```
def set_io_mode(self, graph_name: str, mode: sail.IOMode)->None
```

参数说明:

- graph_name: str

需要配置的计算图的 name。

- mode: sail.IOMode

设置 Engine 的输入/输出 Tensor 所在的内存位置：系统内存或设备内存。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    engine.set_io_mode(graph_name,sail.IOMode.SYSI)
```

5.11.6 get_input_names

获取选定计算图中所有输入 Tensor 的 name

接口形式:

```
def get_input_names(self, graph_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

返回值说明:

- input_names: list

返回选定计算图中所有输入 Tensor 的 name 的列表。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_names = engine.get_input_names(graph_name)
```

5.11.7 get_output_names

获取选定计算图中所有输出 Tensor 的 name。

接口形式:

```
def get_output_names(self, graph_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

返回值说明:

- output_names: list

返回选定计算图中所有输出 Tensor 的 name 的列表。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_names = engine.get_output_names(graph_name)
```

5.11.8 get_max_input_shapes

查询选定计算图中所有输入 Tensor 对应的最大 shape。

在静态模型中，输入 Tensor 的 shape 是固定的，应等于最大 shape。

在动态模型中，输入 Tensor 的 shape 应小于等于最大 shape。

接口形式:

```
def get_max_input_shapes(self, graph_name: str)->dict {str : list}
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

返回值说明:

- max_shapes: dict{str : list}

返回输入 Tensor 中的最大 shape。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    max_input_shapes = engine.get_max_input_shapes(graph_name)
```

5.11.9 get_input_shape

查询选定计算图中特定输入 Tensor 的 shape。

接口形式:

```
def get_input_shape(self, graph_name: str, tensor_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: list

该 name 下的输入 Tensor 中的最大维度的 shape。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    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)
```

5.11.10 get_max_output_shapes

查询选定计算图中所有输出 Tensor 对应的最大 shape。

在静态模型中，输出 Tensor 的 shape 是固定的，应等于最大 shape。

在动态模型中，输出 Tensor 的 shape 应小于等于最大 shape。

接口形式:

```
def get_max_output_shapes(self, graph_name: str)->dict {str : list}
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

返回值说明:

- max_shapes: dict{str : list}

返回输出 Tensor 中的最大 shape。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    max_output_shapes = engine.get_max_output_shapes(graph_name)
```

5.11.11 get_output_shape

查询选定计算图中特定输出 Tensor 的 shape。

接口形式:

```
def get_output_shape(self, graph_name: str, tensor_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: list

该 name 下的输出 Tensor 的 shape。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_name = engine.get_output_names(graph_name)[0]
    output_shape = engine.get_output_shape(graph_name,output_name)
```

5.11.12 get_input_dtype

获取特定计算图的特定输入 Tensor 的数据类型。

接口形式:

```
def get_input_dtype(self, graph_name: str, tensor_name: str)->sail.Dtype
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- datatype: sail.Dtype

返回 Tensor 中数据的数据类型。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_name = engine.get_input_names(graph_name)[0]
    input_dtype = engine.get_input_dtype(graph_name,input_name)
```

5.11.13 get_output_dtype

获取特定计算图的特定输出 Tensor 的数据类型。

接口形式:

```
def get_output_dtype(self, graph_name: str, tensor_name: str)->sail.Dtype
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- datatype: sail.Dtype

返回 Tensor 中数据的数据类型。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_name = engine.get_output_names(graph_name)[0]
    output_dtype = engine.get_output_dtype(graph_name,output_name)
```

5.11.14 get_input_scale

获取特定计算图的特定输入 Tensor 的 scale，只在 int8 模型中有效。

接口形式:

```
def get_input_scale(self, graph_name: str, tensor_name: str)->float32
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- scale: float32

返回 Tensor 数据的 scale。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_name = engine.get_input_names(graph_name)[0]
    input_scale = engine.get_input_scale(graph_name,input_name)
```

5.11.15 get_output_scale

获取特定计算图的特定输出 Tensor 的 scale，只在 int8 模型中有效。

接口形式:

```
def get_output_scale(self, graph_name: str, tensor_name: str)->float32
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- scale: float32

返回 Tensor 数据的 scale。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_name = engine.get_output_names(graph_name)[0]
    output_scale = engine.get_output_scale(graph_name,output_name)
```

5.11.16 process

在特定的计算图上进行前向推理。

接口形式 1:

```
def process(self, graph_name: str, input_tensors: dict {str : numpy.array}, core_list:␣
↪list[int])->dict {str : numpy.array}
    """ Inference with provided system data of input tensors.
```

参数说明 1:

- graph_name: str

特定的计算图 name。

- input_tensors: dict{str : numpy.array}

所有的输入 Tensor 的数据，利用系统内存中的 numpy.array 传入。

- core_list: list[int]

输入参数。该参数仅对支持多核推理的处理器有效，可以选择推理时使用的 core。设 bmodel 为对应的核数为 N，此时 core_list 为空或者 core_list 的长度大于 N，都会使用从 core0 开始的 N 个 core 来做推理。对于仅支持单核推理的处理器可忽略此参数。

返回值说明 1:

- output_tensors: dict{str : numpy.array}

所有的输出 Tensor 的数据，返回类型为 numpy.array 的数据。

接口形式 2:

```
def process(self, graph_name: str, input_tensors: dict {str : sail.Tensor}, output_tensors:␣
↪dict {str : sail.Tensor}, core_list: list[int])->None

def process(self, graph_name: str, input_tensors: dict {str : sail.Tensor}, input_shapes: dict
↪{str : list}, output_tensors: dict {str : sail.Tensor}, core_list: list[int])->None
```

参数说明 2:

- graph_name: str

输入参数。特定的计算图 name。

- input_tensors: dict{str : sail.Tensor}

输入参数。所有的输入 Tensor 的数据，利用 sail.Tensor 传入。

- input_shapes : dict {str : list}

输入参数。所有传入 Tensor 的 shape。

- output_tensors: dict{str : sail.Tensor}

输出参数。所有的输出 Tensor 的数据，利用 sail.Tensor 返回。

- core_list: list[int]

输入参数。该参数仅对支持多核推理的处理器有效，可以选择推理时使用的 core。设 bmodel 为对应的核数为 N，若 core_list 为空则使用从 core0 开始的 N 个 core 做推理；若 core_list 的长度大于 N，则使用 core_list 中对应的前 N 个 core 做推理。对于仅支持单核推理的处理器可忽略此参数。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    # prepare tensor map
    input_tensors_map = engine.create_input_tensors_map(graph_name)
```

(下页继续)

(续上页)

```
    # inference type1
    output_tensors_map = engine.process(graph_name, input_tensors_map)

    # inference type2
    output_tensors_map_ = engine.create_output_tensors_map(graph_name)
    engine.process(graph_name, input_tensors_map, output_tensors_map_)
```

5.11.17 get_device_id

获取 Engine 中的设备 id 号

接口形式:

```
def get_device_id(self)->int
```

返回值说明:

- tpu_id : int

返回 Engine 中的设备 id 号。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    dev_id = engine.get_device_id()
```

5.11.18 create_input_tensors_map

创建输入 Tensor 的映射，在 python 接口中为字典 dict{str : Tensor}

接口形式:

```
def create_input_tensors_map(self, graph_name: str, create_mode: int = -1)->dict{str :␣
↪Tensor}
```

参数说明:

- graph_name: str

特定的计算图 name。

- create_mode: int

创建 Tensor 分配内存的模式。为 0 时只分配系统内存，为 1 时只分配设备内存，其他时则根据 Engine 中 IOMode 的配置分配。

返回值说明:

output: dict{str : Tensor}

返回 name:tensor 的字典。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    # prepare tensor map
    input_tensors_map = engine.create_input_tensors_map(graph_name)
```

5.11.19 create_output_tensors_map

创建输入 Tensor 的映射，在 python 接口中为字典 dict{str : Tensor}

接口形式:

```
def create_output_tensors_map(self, graph_name: str, create_mode: int = -1)->dict{str :
↪Tensor}
```

参数说明:

- graph_name: str

特定的计算图 name。

- create_mode: int

创建 Tensor 分配内存的模式。为 0 时只分配系统内存，为 1 时只分配设备内存，其他时则根据 Engine 中 IOMode 的配置分配。

返回值说明:

output: dict{str : Tensor}

返回 name:tensor 的字典。

示例代码:

```
import sophon.sail as sail


if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
```

(下页继续)

(续上页)

```
    # prepare tensor map
    output_tensors_map = engine.create_output_tensors_map(graph_name)
```

5.12 sail.MultiEngine

多线程的推理引擎，实现特定计算图的多线程推理。

5.12.1 MultiEngine

初始化 MutiEngine。

接口形式:

```
def __init__(self, bmodel_path: str, device_ids: list[int], sys_out: bool=True, graph_idx:␣
↪int=0)
```

参数说明:

- bmodel_path: str

bmodel 所在的文件路径。

- device_ids: lists[int]

该 MultiEngine 可见的智能视觉深度学习处理器的 ID。

- sys_out: bool

表示是否将结果拷贝到系统内存，默认为 True

- graph_idx : int

特定的计算图的 index。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
```

5.12.2 set_print_flag

设置是否打印调试信息。

接口形式:

```
def set_print_flag(self, print_flag: bool)->None
```

参数说明:

- print_flag: bool

为 True 时，打印调试信息，否则不打印。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    engine.set_print_flag(True)
```

5.12.3 set_print_time

设置是否打印主要处理耗时。

接口形式:

```
def set_print_time(self, print_flag: bool)->None
```

参数说明:

- print_flag: bool

为 True 时，打印主要耗时，否则不打印。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    engine.set_print_time(True)
```

5.12.4 get_device_ids

获取 MultiEngine 中所有可用的智能视觉深度学习处理器的 id。

接口形式:

```
def get_device_ids(self) -> list[int]
```

返回值说明:

- device_ids: list[int]

返回可见的智能视觉深度学习处理器的 ids

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    ids = engine.get_device_ids()
```

5.12.5 get_graph_names

获取 MultiEngine 中所有载入的计算图的名称。

接口形式:

```
def get_graph_names(self)->list
```

返回值说明:

- graph_names: list

MultiEngine 中所有计算图的 name 的列表。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
```

5.12.6 get_input_names

获取选定计算图中所有输入 Tensor 的 name

接口形式:

```
def get_input_names(self, graph_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

返回值说明:

- input_names: list

返回选定计算图中所有输入 Tensor 的 name 的列表。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    input_names = engine.get_input_names(graph_names[0])
```

5.12.7 get_output_names

获取选定计算图中所有输出 Tensor 的 name。

接口形式:

```
def get_output_names(self, graph_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

返回值说明:

- output_names: list

返回选定计算图中所有输出 Tensor 的 name 的列表。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    output_names = engine.get_output_names(graph_names[0])
```

5.12.8 get_input_shape

查询选定计算图中特定输入 Tensor 的 shape。

接口形式:

```
def get_input_shape(self, graph_name: str, tensor_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: list

该 name 下的输入 Tensor 中的最大维度的 shape。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    input_names = engine.get_input_names(graph_names[0])
    input_shape = engine.get_input_shape(graph_name,input_names[0])
```

5.12.9 get_output_shape

查询选定计算图中特定输出 Tensor 的 shape。

接口形式:

```
def get_output_shape(self, graph_name: str, tensor_name: str)->list
```

参数说明:

- graph_name: str

设定需要查询的计算图的 name。

- tensor_name: str

需要查询的 Tensor 的 name。

返回值说明:

- tensor_shape: list

该 name 下的输出 Tensor 的 shape。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    output_names = engine.get_output_names(graph_names[0])
    output_shape = engine.get_output_shape(graph_name,output_names[0])
```

5.12.10 process

在特定的计算图上进行推理，需要提供系统内存的输入数据。

接口形式:

```
def process(self, input_tensors: dict {str : numpy.array})->dict {str : numpy.array}

def process(self, input_tensors: list[dict{str: sophon.sail.Tensor}] )->dict {str : Tensor}
```

参数说明:

- input_tensors: dict{ str : numpy.array }

输入的 Tensors。

返回值说明:

- output_tensors: dict{str : numpy.array}

返回推理之后的结果。

示例代码:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = [0,1]
    handle = sail.Handle(0)
    bmodel_path = 'your_bmodel.bmodel'
    engine = sail.MultiEngine(bmodel_path, dev_id)
    graph_name = engine.get_graph_names()[0]
    input_names = engine.get_input_names(graph_name)
    output_names = engine.get_output_names(graph_name)

    input_tensors_map = {}

    # form 1
    input_numpy_map = {}
    for input_name in input_names:
        data = np.ones(engine.get_input_shape(graph_name,input_name),dtype=np.float32)
        input_numpy_map = {input_name:data}
    output_tensors_map = engine.process(input_numpy_map)
    print(output_tensors_map)

    # form 2
    for input_name in input_names:
        data = np.ones(engine.get_input_shape(graph_name,input_name),dtype=np.float32)
        tensor = sail.Tensor(handle,data)
        input_tensors_map[input_name] = tensor
    input_tensors_vector = [input_tensors_map]
    output_tensors_map = engine.process(input_tensors_vector)
    print(output_tensors_map)
```

5.13 sail.bm_image

bm_image 是 BMCV 中的基本结构，封装了一张图像的主要信息，是后续 BMImage 和 BMImageArray 的内部元素。

接口形式:

```
def width(self) -> int
```

返回值说明:

- width：int

返回图像的宽。

接口形式:

```
def height(self) -> int
```

返回值说明:

- height：int

返回图像的高。

接口形式:

```
def format(self) -> sail.Format
```

返回值说明:

- format：sail.Format

返回图像的格式。

接口形式:

```
def dtype(self) -> sail.ImgDtype
```

返回值说明:

- dtype：sail.ImgDtype

返回图像的数据格式。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image.jpg"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmimg = BMimg.data()# here is a sail.bm_image
    print(bmimg.width(),bmimg.height(),bmimg.format(),bmimg.dtype())
```

5.14 sail.BMImage

BMImage 封装了一张图片的全部信息，可利用 Bmcv 接口将 BMImage 转换为 Tensor 进行模型推理。

BMImage 也是通过 Bmcv 接口进行其他图像处理操作的基本数据类型。

5.14.1 __init__

初始化 BMImage。

接口形式:

```
def __init__(self)

def __init__(self, handle: sail.Handle, h: int, w: int, format: sail.Format, dtype: sail.
↪ImgDtype)
```

参数说明:

- handle: sail.Handle

设定 BMImage 所在的设备句柄。

- h: int

图像的高。

- w: int

图像的宽。

- format：sail.Format

图像的格式。

- dtype: sail.ImgDtype

图像的数据类型。

5.14.2 width

获取图像的宽。

接口形式:

```
def width(self)->int
```

返回值说明:

- width：int

返回图像的宽。

5.14.3 height

获取图像的高。

接口形式:

```
def height(self)->int
```

返回值说明:

- height：int

返回图像的高。

5.14.4 format

获取图像的格式。

接口形式:

```
def format(self) -> sail.Format
```

返回值说明:

- format：sail.Format

返回图像的格式。

5.14.5 dtype

获取图像的数据类型。

接口形式:

```
def dtype(self)->sail.ImgDtype
```

返回值说明:

- dtype: sail.ImgDtype

返回图像的数据类型。

5.14.6 data

获取 BMImage 内部的 bm_image。

接口形式:

```
def data(self) -> sail.bm_image
```

返回值说明:

- img : sail.bm_image

返回图像内部的 bm_image。

5.14.7 get_device_id

获取 BMImage 中的设备 id 号。

接口形式:

```
def get_device_id(self) -> int
```

返回值说明:

- device_id : int

返回 BMImage 中的设备 id 号

5.14.8 get_handle

获取 BMImage 中的 Handle。

接口形式:

```
def get_handle(self):
```

返回值说明:

- Handle : Handle

返回 BMImage 中的 Handle

5.14.9 asmat

将 BMImage 中的数据转换成 numpy.ndarray

接口形式:

```
def asmat(self) -> numpy.ndarray[numpy.uint8]
```

返回值说明:

- image : numpy.ndarray[numpy.uint8]

返回 BMImage 中的数据。

5.14.10 get_plane_num

获取 BMImage 中图像 plane 的数量。

接口形式:

```
def get_plane_num(self) -> int:
```

返回值说明:

- planes_num : int

返回 BMImage 中图像 plane 的数量。

5.14.11 align

将 BMImage 64 对齐

接口形式:

```
def align(self) -> int:
```

返回值说明:

- ret : int

返回 BMImage 是否对齐成功,-1 代表失败,0 代表成功

5.14.12 check_align

获取 BMImage 中图像是否对齐

接口形式:

```
def check_align(self) -> bool:
```

返回值说明:

- ret : bool

1 代表已对齐,0 代表未对齐

5.14.13 unalign

将 BMImage 不对齐

接口形式:

```
def unalign(self) -> int:
```

返回值说明:

- ret：int

返回 BMImage 是否不对齐成功,-1 代表失败,0 代表成功

5.14.14 check_contiguous_memory

获取 BMImage 中图像内存是否连续

接口形式:

```
def check_contiguous_memory(self) -> bool:
```

返回值说明:

- ret：bool

1 代表连续,0 代表不连续

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    file_path = '/data/jinyu.lu/jpu_test/1920x1080_yuvj420.jpg' # 请替换为您的文件路径
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, False, dev_id)
    BMimg = sail.BMImage()
    ret = decoder.read(handle, BMimg)

    # get bm_image
    bm_image = BMimg.data()

    # get BMimg width,height,dtype,format,device_id,plane_num,handle
    print(BMimg.width(), BMimg.height(), BMimg.format(), BMimg.dtype(), BMimg.get_
↪device_id(), BMimg.get_plane_num(), BMimg.get_handle())

    # get mat
    np_data = BMimg.asmat()

    # align BMimg
    ret = BMimg.align()
    if ret:
        print("align success")
    else:
        print("align failed")

    print(BMimg.check_align())

    # unalign BMimg
    ret = BMimg.unalign()
    if ret:
```

(下页继续)

(续上页)

```
    print("unalign success")
else:
    print("unalign failed")

# check contiguous memory
print(BMimg.check_contiguous_memory())
```

5.15 sail.BMImageArray

BMImageArray 是 BMImage 的数组，可为多张图片申请连续的内存空间。

在声明 BMImageArray 时需要根据图片数量指定不同的实例

例：4 张图片时 BMImageArray 的构造方式如：images = BMImageArray4D()

5.15.1 __init__

初始化 BMImageArray。

接口形式:

```
def __init__(self) :

def __init__(self, handle: sail.Handle, h: int, w: int, format: sail.Format, dtype: sail.
↪ImgDtype)
```

参数说明:

- handle: sail.Handle

设定 BMImage 所在的设备句柄。

- h: int

图像的高。

- w: int

图像的宽。

- format：sail.Format

图像的格式。

- dtype: sail.ImgDtype

图像的数据类型。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D
    images = sail.BMImageArray4D()

    # Create BMImageArray4D with parameters
    handle = sail.Handle(1)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,1)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
```

5.15.2 __getitem__

获取 index 上的 bm_image。

接口形式:

```
def __getitem__(self, i: int)->sail.bm_image
```

参数说明:

- i: int

需要返回图像的 index。

返回值说明:

- img: sail.bm_image

返回 index 上的图像。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    handle = sail.Handle(1)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,1)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())

    images.copy_from(0,ori_img)
    # get the bm_image from index 0
    img_0 = images.__getitem__(0)
    print("image0 from bmimg_array:",img_0.width(),img_0.height(),img_0.dtype())
```

5.15.3 __setitem__

将图像拷贝到特定的索引上。

接口形式:

```
def __setitem__(self, i: int, data: sail.bm_image)->None
```

参数说明:

- i: int

输入需要拷贝到的 index

- data: sail.bm_image

需要拷贝的图像数据。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    handle = sail.Handle(1)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,1)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # copy image to the specified index
    images.__setitem__(3,ori_img.data())
```

5.15.4 copy_from

将图像拷贝到特定的索引上。

接口形式:

```
def copy_from(self, i: int, data: sail.BMImage)->None
```

参数说明:

- i: int

输入需要拷贝到的 index

- data: sail.BMImage

需要拷贝的图像数据。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    handle = sail.Handle(1)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,1)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # copy image to the specified index
    images.copy_from(0,ori_img)
```

5.15.5 attach_from

将图像 attach 到特定的索引上，这里没有内存拷贝，所以需要原始数据已经被缓存。

接口形式:

```
def attach_from(self, i: int, data: BMImage)->None
```

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    handle = sail.Handle(1)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,1)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # Attach image to the specified index
    images.attach_from(1,ori_img)
```

5.15.6 get_device_id

获取 BMImageArray 中的设备号。

接口形式:

```
def get_device_id(self) -> int:
```

返回值说明:

- device_id: int

BMImageArray 中的设备 id 号

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    handle = sail.Handle(1)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,1)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # Get device id of this BMImageArray
    devid = images.get_device_id()
    print("device id:",devid)
```

5.16 sail.Decoder

解码器，可实现图像或视频的解码。

图像解码像素格式支持说明:

- 硬解支持 jpeg baseline，不是所有的 jpeg
- 视频支持硬解 h264,h265。输出的像素格式为 YUV-nv12、YUVJ420P 或者 YUV420P；

5.16.1 __init__

初始化 Decoder。

接口形式:

```
def __init__(self, file_path: str, compressed: bool=True, tpu_id: int=0)
```

参数说明:

- file_path: str

图像或视频文件的 Path 或 RTSP 的 URL。

- compressed: bool

是否将解码的输出压缩为 NV12，default：True。开启之后可以节省内存、节省带宽，但是输入视频必须要满足宽能被 16 整除才行，且输入必须为视频时才能生效。

- tpu_id: int

设置使用的智能视觉深度学习处理器 id 号。

5.16.2 is_opened

判断源文件是否打开。

接口形式:

```
def is_opened(self) -> bool
```

返回值说明:

- judge_ret: bool

打开成功返回 True，失败返回 False。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'  # 请替换为您的文件路径
    dev_id = 0
    decoder = sail.Decoder(file_path, True, dev_id)
    ret = decoder.is_opened()
    print("Decoder opened:", ret)
```

5.16.3 read

从 Decoder 中读取一帧图像。

接口形式 1:

```
def read(self, handle: sail.Handle, image: sail.BMImage)->int
```

参数说明 1:

- handle: sail.Handle

输入参数。Decoder 使用的智能视觉深度学习处理器的 Handle。

- image: sail.BMImage

输出参数。将数据读取到 image 中。

返回值说明 1:

- judge_ret: int

读取成功返回 0，失败返回其他值。

接口形式 2:

```
def read(self, handle: sail.Handle)->sail.BMImage
```

参数说明 2:

- handle: sail.Handle

输入参数。Decoder 使用的智能视觉深度学习处理器的 Handle。

返回值说明 2:

- image: sail.BMImage

将数据读取到 image 中。

示例代码 1:

```
import sophon.sail as sail

if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'  # 请替换为您的文件路径
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, True, dev_id)
    image = sail.BMImage()
    ret = decoder.read(handle, image)
    if ret == 0:
        print("Frame read successfully")
    else:
        print("Failed to read frame")
```

示例代码 2:

```
import sophon.sail as sail

if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'  # 请替换为您的文件路径
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, True, dev_id)
    BMimg = decoder.read(handle)
```

5.16.4 read_

从 Decoder 中读取一帧图像。

接口形式:

```
def read_(self, handle: sail.Handle, image: sail.bm_image)->int
```

参数说明:

- handle: sail.Handle

输入参数。Decoder 使用的智能视觉深度学习处理器的 Handle。

- image: sail.bm_image

输出参数。将数据读取到 image 中。

返回值说明:

- judge_ret: int

读取成功返回 0，失败返回其他值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'  # 请替换为您的文件路径
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, True, dev_id)
    image = sail.BMImage()
    bm_img = image.data()
    ret = decoder.read_(handle, bm_img)
    if ret == 0:
        print("Frame read successfully into bm_image")
    else:
        print("Failed to read frame into bm_image")
```

5.16.5 get_frame_shape

获取 Decoder 中 frame 中的 shape。

接口形式:

```
def get_frame_shape(self)->list
```

返回值说明:

- frame_shape: list

返回当前 frame 的 shape。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'  # 请替换为您的文件路径
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path,True,dev_id)
    print(decoder.get_frame_shape())
```

5.16.6 release

释放 Decoder 资源。

接口形式:

```
def release(self) -> None
```

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = 'your_video_file_path.mp4'  # 请替换为您的文件路径
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path,True,dev_id)
    decoder.release()
```

5.16.7 reconnect

Decoder 再次连接。

接口形式:

```
def reconnect(self) -> None
```

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
    # 重新连接解码器
    decoder.reconnect()
```

5.16.8 enable_dump

开启解码器的 dump 输入视频功能（不经编码），并缓存最多 1000 帧未解码的视频。

接口形式:

```
def enable_dump(dump_max_seconds: int):
```

参数说明:

- dump_max_seconds: int

输入参数。dump 视频的最大时长，也是内部 AVpacket 缓存队列的最大长度。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
    dump_max_seconds = 100
    decoder.enable_dump(dump_max_seconds)
```

5.16.9 disable_dump

关闭解码器的 dump 输入视频功能，并清空开启此功能时缓存的视频帧

接口形式:

```
def disable_dump():
    """ Disable  input video dump without encode.
    """
```

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
    decoder.enable_dump(100)
    decoder.disable_dump()
```

5.16.10 dump

在调用此函数的时刻，dump 下前后数秒的输入视频。由于未经编码，必须 dump 下前后数秒内所有帧所依赖的关键帧。因而接口的 dump 实现以 gop 为单位，实际 dump 下的视频时长将高于输入参数时长。误差取决于输入视频的 gop_size，gop 越大，误差越大。

接口形式:

```
def dump(dump_pre_seconds: int, dump_post_seconds: int, file_path: str)->int
```

- dump_pre_seconds: int

输入参数。保存调用此接口时刻之前的数秒视频。

- dump_post_seconds: int

输入参数。保存调用此接口时刻之后的数秒视频。

- file_path: str

输入参数。视频路径。

返回值说明:

- judge_ret: int

成功返回 0, 失败返回其他值。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
    dump_pre_seconds = 3
    dump_post_seconds = 3
    output_file_path = "output_video_path.mp4"
    int ret = decoder.dump(dump_pre_seconds, dump_post_seconds, output_file_path)
    if ret == 0:
        print("successfully!")
    else:
        print("Failed!")
```

5.17 sail.Encoder

编码器，可实现图像或视频的编码，以及保存视频文件、推 rtsp/rtmp 流。

5.17.1 __init__

初始化 Encoder。

图片编码器初始化:

图片编码器:

```
def __init__(self)
```

视频编码器初始化:

视频编码器接口形式 1:

```
def __init__(self, output_path: str, handle: sail.Handle, enc_fmt: str, pix_fmt: str, enc_
↪params: str, cache_buffer_length: int=5, abort_policy: int=0)
```

视频编码器接口形式 2:

```
def __init__(self, output_path: str, device_id: int, enc_fmt: str, pix_fmt: str, enc_
↪params: str, cache_buffer_length: int=5, abort_policy: int=0)
```

参数说明:

- output_path: str

输入参数。编码视频输出路径，支持本地文件（MP4，ts 等）和 rtsp/rtmp 流。

- handle: sail.Handle

输入参数。编码器 handle 实例。(与 device_id 二选一)

- device_id: int

输入参数。编码器 device_id。(与 handle 二选一，指定 device_id 时，编码器内部将会创建 Handle)

- enc_fmt: str

输入参数。编码格式，支持 h264_bm 和 h265_bm/hevc_bm。

- pix_fmt: str

输入参数。编码输出的像素格式，支持 NV12 和 I420。推荐使用 I420。

- enc_params: str

输入参数。编码参数，如 "width=1920:height=1080:gop=32:gop_preset=3:framerate=25:bitrate=2000"，其中 width 和 height 是必须的，默认用 bitrate 控制质量，单位为 kbps，参数中指定 qp 时 bitrate 失效。

- cache_buffer_length: int

输入参数。内部缓存队列长度，默认为 5。sail.Encoder 内部会维护一个缓存队列，从而在推流时提升流控容错。

- abort_policy: int

输入参数。缓存队列已满时，video_write 接口的拒绝策略。设为 0 时，video_write 接口立即返回-1。设为 1 时，pop 队列头。设为 2 时，清空队列。设为 3 时，阻塞直到编码线程消耗一帧，队列产生空位。

5.17.2 is_opened

判断编码器是否打开。

接口形式:

```
def is_opened(self) -> bool
```

返回值说明:

- judge_ret: bool

编码器打开返回 True，失败返回 False。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    out_path = "path/to/your/output/file"
```

(下页继续)

(续上页)

```
    enc_fmt = "h264_bm"
    pix_fmt = "I420"
    enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25"
    cache_buffer_length = 5
    abort_policy = 0
    encoder = sail.Encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy)
    print(encoder.is_opened())
```

5.17.3 pic_encode

编码一张图片，并返回编码后的 data。

接口形式 1:

```
def pic_encode(self, ext: str, image: BMImage)->numpy.array
```

接口形式 2:

```
def pic_encode(self, ext: str, image: bm_image)->numpy.array
```

参数说明:

- ext: str

输入参数。图片编码格式。".jpg" , ".png" 等。

- image: BMImage/bm_image

输入参数。输入图片，只支持 FORMAT_BGR_PACKED，DATA_TYPE_EXT_1N_BYTE 的图片。

返回值说明:

- data: numpy.array

编码后放在系统内存中的数据。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    img_path = "path/to/your/output/file"
    decoder = sail.Decoder(img_path,False,dev_id)
    img = decoder.read(handle)
    # img = decoder.read(handle).data()  //bm_image
    encoder = sail.Encoder()
    data = encoder.pic_encode(".jpg", img)
    print(data)
```

5.17.4 video_write

向视频编码器送入一帧图像。异步接口，做格式转换后，放入内部的缓存队列中。

接口形式 1:

```
def video_write(self, image: sail.BMImage)->int
```

接口形式 2:

```
def video_write(self, image: sail.bm_image)->int
```

参数说明:

- image: sail.BMImage

输入参数。输入图片。

在 BM1684 上，当编码器像素格式（即 pix_fmt）为 I420 时，待编码的 image 的 shape 可以与编码器的宽高不同；当像素格式为 NV12 时，要求 image 的 shape 与编码器的宽高一致，内部使用 bmcv_image_storage_convert 做格式转换，可能占用 NPU 资源。

在 BM1684X 上，待编码的 image 的 shape 可以与编码器的宽高不同，内部使用 bmcv_image_vpp_convert 做 resize 和格式转换。

返回值说明:

- judge_ret: int

成功返回 0，内部缓存队列已满返回-1。内部缓存队列中有一帧编码失败时返回-2。有一帧成功编码，但推流失败返回-3。未知的拒绝策略返回-4。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    img_path = "your_img_path"
    decoder = sail.Decoder(img_path,False,dev_id)
    img = decoder.read(handle)
    out_path = "path/to/your/output/file"
    enc_fmt = "h264_bm"
    pix_fmt = "I420"
    enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25"
    cache_buffer_length = 5
    abort_policy = 0
    encoder = sail.Encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy)
    ret = encoder.video_write(img)
    # ret = encoder.video_write(img.data()) # sail.bm_image
    print(ret)
```

5.17.5 release

释放编码器。

接口形式:

```
def release(self)->None
```

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    out_path = "path/to/your/output/file"
    enc_fmt = "h264_bm"
    pix_fmt = "I420"
    enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25"
    cache_buffer_length = 5
    abort_policy = 0
    encoder = sail.Encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy)
    encoder.release()
```

5.18 sail.Decoder_RawStream

裸流解码器，可实现 H264/H265 的解码。

5.18.1 __init__

接口形式:

```
def __init__(self, tpu_id: int, decformat: str)
```

参数说明:

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id，默认为 0。

- decformat: str

输入参数。输入图像的格式，支持 h264 和 h265

5.18.2 read

从 Decoder 中读取一帧图像。

接口形式 1:

```
def read(self, data: bytes, image: BMImage, continue_frame: bool = False) -> int
```

参数说明 1:

- data: bytes

输入参数。裸流的二进制数据。

- image: sail.BMImage

输出参数。将数据读取到 image 中。

- continue_frame: bool

输入参数。是否连续读帧, 默认为 False。

返回值说明 1:

- judge_ret: int

读取成功返回 0，失败返回其他值。

5.18.3 read_

从 Decoder 中读取一帧图像。

接口形式 1:

```
read_(self, data_bytes: bytes, image: bm_image, continue_frame: bool = False)
```

参数说明 1:

- data: bytes

输入参数。裸流的二进制数据。

- image: sail.bm_image

输出参数。将数据读取到 image 中。

- continue_frame: bool

输入参数。是否连续读帧, 默认为 False。

返回值说明 1:

- judge_ret: int

读取成功返回 0，失败返回其他值。

5.18.4 release

释放 Decoder 资源。

接口形式:

```
def release(self) -> None
```

示例代码:

```
import sophon.sail as sail

filepath = 'car.264'
with open(filepath, 'rb') as f:
    raw264 = f.read()

decoder = sail.Decoder_RawStream(0, 'h264')
encoder = sail.Encoder('output.mp4', 0, 'h264_bm', 'I420',
                  'width=1920:height=1080:bitrate=3000')
for i in range(500):
    bmi = sail.BMImage()
    ret = decoder.read(raw264, bmi, True)
    encoder.video_write(bmi)
```

5.19 sail.Bmcv

Bmcv 封装了常用的图像处理接口, 支持硬件加速。

实现接口硬件说明

本文档中, 在 BM1684/BM1684X 上实现的接口, 负责其实现的硬件单元可能有不同情况 (i.e. crop_and_resize 在 BM1684 上由 VPP+ 智能视觉深度学习处理器实现)。影响实现的硬件单元的因素如下:

1. 输入图片/输出图片数量不大于 16
2. 图片无需按照原图比例进行缩放
3. 输入图片/输出图片不为以下数据格式: DATA_TYPE_EXT_1N_BYTE_SIGNED DATA_TYPE_EXT_4N_BYTE DATA_TYPE_EXT_FLOAT32
4. 输入的图片 device memory 不在 DDR0 上
5. 输入图片格式不为 FORMAT_YUV422P, 输出图片格式不为 FORMAT_NV12 或 FORMAT_COMPRESSED, 并且输入图片格式-输出图片格式不为以下组合:

input_format	output_format
FORMAT_RGBP_SEPARATE	FORMAT_ARGB_PACKED FORMAT_ABGR_PACKED
FORMAT_BGRP_SEPARATE	FORMAT_ARGB_PACKED FORMAT_ABGR_PACKED
FORMAT_GRAY	FORMAT_YUV420P FORMAT_YUV422P FORMAT_YUV444P
FORMAT_YUV420P	FORMAT_GRAY FORMAT_YUV422P FORMAT_YUV444P
FORMAT_YUV444P	FORMAT_GRAY
FORMAT_COMPRESSED	FORMAT_GRAY FORMAT_YUV422P FORMAT_YUV444P

当且仅当满足上述 5 个条件时, 接口实现硬件为" VPP+ 智能视觉深度学习处理器" 的 Bmcv 接口会使用 VPP; 否则, 将会使用智能视觉深度学习处理器, 智能视觉深度学习处理器仅支持最邻近插值（Nearest Interpolitan）, 如不满足上述使用 VPP 的条件且缩放算法为其它插值策略, 将会报错。

5.19.1 __init__

初始化 Bmcv

接口形式:

```
def __init__(self, handle: sail.Handle)
```

参数说明:

- handle: sail.Handle

指定 Bmcv 使用的设备句柄。

5.19.2 bm_image_to_tensor

将 BMImage/BMImageArray 转换为 Tensor。

接口形式 1:

```
def bm_image_to_tensor(self, image: sail.BMImage) -> sail.Tensor
```

参数说明 1:

- image: sail.BMImage

需要转换的图像数据。

返回值说明 1:

- tensor: sail.Tensor

返回转换后的 Tensor。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
```

接口形式 2:

```
def bm_image_to_tensor(self,
        image: sail.BMImageArray,
        tensor) -> None
```

参数说明 2:

- image: sail.BMImageArray

输入参数。需要转换的图像数据。

- tensor: sail.Tensor

输出参数。转换后的 Tensor。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = sail.Tensor(handle,(1920,1080),sail.Dtype.BM_FLOAT32,True,True)
    bmcv.bm_image_to_tensor(BMimg,tensor)
```

5.19.3 tensor_to_bm_image

将 Tensor 转换为 BMImage/BMImageArray。

接口形式 1:

```
def tensor_to_bm_image(self,
        tensor: sail.Tensor,
        bgr2rgb: bool=False) -> sail.BMImage
```

参数说明 1:

- tensor: sail.Tensor

输入参数。待转换的 Tensor。

- bgr2rgb: bool, default: False

输入参数。是否进行图像的通道变换。

返回值说明 1:

- image : sail.BMImage

返回转换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
    BMimg2 = bmcv.tensor_to_bm_image(tensor)
```

接口形式 2:

```
def tensor_to_bm_image(self,
        tensor: sail.Tensor,
        img: sail.BMImage | sail.BMImageArray,
        bgr2rgb: bool=False) -> None
```

参数说明 2:

- tensor: sail.Tensor

输入参数。待转换的 Tensor。

- img : sail.BMImage | sail.BMImageArray

输出参数。返回转换后的图像。

- bgr2rgb: bool, default: False

输入参数。是否进行图像的通道变换。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
    BMimg2 = sail.BMImage()
    bmcv.tensor_to_bm_image(tensor,BMimg2)
```

5.19.4 crop_and_resize

对图片进行裁剪并 resize。

实现硬件 * BM1684: VPP+ 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def crop_and_resize(self,
        input: sail.BMImage | sail.BMImageArray,
        crop_x0: int,
        crop_y0: int,
        crop_w: int,
        crop_h: int,
        resize_w: int,
        resize_h: int,
        resize_alg: sail.bmcv_resize_algorithm=sail.bmcv_resize_algorithm.BMCV_INTER_
↪NEAREST)
            -> sail.BMImage
```

参数说明:

- input : sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- crop_x0 : int

裁剪窗口在 x 轴上的起始点。

- crop_y0 : int

裁剪窗口在 y 轴上的起始点。

- crop_w : int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- resize_alg：sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg3 = bmcv.crop_and_resize(BMimg,0,0,BMimg.width(),BMimg.height(),640,640,
↪sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST)
```

5.19.5 crop

对图像进行裁剪。

接口形式 1:

```
def crop(self,
        input: sail.BMImage | sail.BMImageArray,
        crop_x0: int,
        crop_y0: int,
        crop_w: int,
        crop_h: int) -> sail.BMImage | sail.BMImageArray
```

参数说明:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- crop_x0 : int

裁剪窗口在 x 轴上的起始点。

- crop_y0 : int

裁剪窗口在 y 轴上的起始点。

- crop_w : int

裁剪窗口的宽。

- crop_h : int

裁剪窗口的高。

返回值说明:

- output : sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0, crop_y0, crop_w, crop_h = 100, 100, 200, 200
    cropped_BMimg = bmcv.crop(BMimg, crop_x0, crop_y0, crop_w, crop_h)
```

对图像进行裁剪, 可从一张图上裁剪出多个小图

接口形式 2:

```
def crop(self,
        input: sail.BMImage
        rects: list[list[]] ) -> list[sail.BMImage]
```

参数说明:

- input : sail.BMImage

待处理的图像。

- rects: list[list[]]

[[crop_x0,crop_y0,crop_w0,crop_h0],[crop_x1,crop_y1,crop_w1,crop_h1]]

- crop_xi : int

第 i 个裁剪窗口在 x 轴上的起始点。

- crop_yi : int

第 i 个裁剪窗口在 y 轴上的起始点。

- crop_wi : int

第 i 个裁剪窗口的宽。

- crop_hi : int

第 i 个裁剪窗口的高。

返回值说明 1:

- output : list[sail.BMImage]

返回处理后的图像列表。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    rects = [
        [0, 0, 40, 40],
        [40, 40, 80, 80],
        #...more
    ]
    cropped_images_list = bmcv.crop(BMimg, rects)
```

5.19.6 resize

对图像进行 resize。

接口形式:

```
def resize(self,
        input: sail.BMImage | sail.BMImageArray,
        resize_w: int,
        resize_h: int,
        resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)
    -> sail.BMImage | sail.BMImageArray
```

参数说明:

- input : sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- resize_alg：sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg_resize = bmcv.resize(BMimg,640,640,resize_alg=sail.bmcv_resize_algorithm.
↪BMCV_INTER_NEAREST)
```

5.19.7 vpp_crop_and_resize

利用 VPP 硬件加速图片的裁剪与 resize。

实现硬件 * BM1684: VPP * BM1684X: VPP

接口形式:

```
def vpp_crop_and_resize(self,
                input: sail.BMImage | sail.BMImageArray,
                crop_x0: int,
                crop_y0: int,
                crop_w: int,
                crop_h: int,
                resize_w: int,
                resize_h: int,
                resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)
                -> sail.BMImage | sail.BMImageArray
```

参数说明:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- crop_x0 : int

裁剪窗口在 x 轴上的起始点。

- crop_y0 : int

裁剪窗口在 y 轴上的起始点。

- crop_w : int

裁剪窗口的宽。

- crop_h : int

裁剪窗口的高。

- resize_w : int

图像 resize 的目标宽度。

- resize_h : int

图像 resize 的目标高度。

- resize_alg : sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- output : sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0 = 100
    crop_y0 = 100
    crop_w = 200
    crop_h = 200
    resize_w = 300
    resize_h = 300

    resized_BMimg = bmcv.vpp_crop_and_resize(
        BMimg,
        crop_x0,
```

(下页继续)

(续上页)

```
        crop_y0,
        crop_w,
        crop_h,
        resize_w,
        resize_h,
        sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST
    )
```

5.19.8 vpp_crop_and_resize_padding

利用 VPP 硬件加速图片的裁剪与 resize，并 padding 到指定大小。

实现硬件 * BM1684: VPP * BM1684X: VPP

接口形式:

```
def vpp_crop_and_resize_padding(self,
                    input: sail.BMImage | sail.BMImageArray,
                    crop_x0: int,
                    crop_y0: int,
                    crop_w: int,
                    crop_h: int,
                    resize_w: int,
                    resize_h: int,
                    padding: sail.PaddingAtrr,
                    resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)
                     -> sail.BMImage | sail.BMImageArray
```

参数说明:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- padding：sail.PaddingAtrr

padding 的配置信息。

- resize_alg：sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    paddingatt = sail.PaddingAtrr()
    paddingatt.set_stx(0)
    paddingatt.set_sty(0)
    paddingatt.set_w(640)
    paddingatt.set_h(640)
    paddingatt.set_r(114)
    paddingatt.set_g(114)
    paddingatt.set_b(114)
    BMimg4 = bmcv.vpp_crop_and_resize_padding(BMimg,0,0,BMimg.width(),BMimg.
↪height(),640,640,paddingatt)
```

5.19.9 vpp_crop

利用 VPP 硬件加速图片的裁剪。

接口形式:

```
def vpp_crop(self,
    input: sail.BMImage | sail.BMImageArray,
    crop_x0: int,
    crop_y0: int,
    crop_w: int,
    crop_h: int)->sail.BMImage | sail.BMImageArray
```

参数说明:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

返回值说明:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0 = 100
    crop_y0 = 100
    crop_w = 200
    crop_h = 200
    BMimg4 = bmcv.vpp_crop(BMimg,crop_x0,crop_y0,crop_w,crop_h)
```

5.19.10 vpp_resize

利用 VPP 硬件加速图片的 resize，采用最近邻插值算法。

接口形式 1:

```
def vpp_resize(self,
        input: sail.BMImage | sail.BMImageArray,
        resize_w: int,
        resize_h: int,
        resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)->sail.BMImage |␣
↪sail.BMImageArray
```

参数说明 1:

· input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

· resize_w：int

图像 resize 的目标宽度。

· resize_h：int

图像 resize 的目标高度。

· resize_alg：sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明 1:

· output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg_resize = bmcv.vpp_resize(BMimg,640,640,resize_alg=sail.bmcv_resize_
↪algorithm.BMCV_INTER_NEAREST)
```

接口形式 2:

```
def vpp_resize(self,
       input: sail.BMImage | sail.BMImageArray,
       output: sail.BMImage | sail.BMImageArray,
       resize_w: int,
       resize_h: int,
       resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)->None
```

参数说明 2:

· input：sail.BMImage | sail.BMImageArray

输入参数。待处理的图像或图像数组。

· output：sail.BMImage | sail.BMImageArray

输出参数。处理后的图像或图像数组。

· resize_w：int

输入参数。图像 resize 的目标宽度。

- resize_h : int

输入参数。图像 resize 的目标高度。

- resize_alg : sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg_resize = sail.BMImage()
    bmcv.vpp_resize(BMimg,BMimg_resize,640,640,resize_alg=sail.bmcv_resize_algorithm.
↪BMCV_INTER_NEAREST)
```

5.19.11 vpp_resize_padding

利用 VPP 硬件加速图片的 resize，并 padding。

接口形式:

```
def vpp_resize_padding(self,
        input: sail.BMImage | sail.BMImageArray,
        resize_w: int,
        resize_h: int,
        padding: PaddingAtrr,
        resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)->sail.BMImage |␣
↪sail.BMImageArray
```

参数说明:

- input : sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- resize_w : int

图像 resize 的目标宽度。

- resize_h : int

图像 resize 的目标高度。

- padding : sail.PaddingAtrr

padding 的配置信息。

- resize_alg：sail.bmcv_resize_algorithm

图像 resize 的插值算法，默认为 sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

返回值说明:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name, True, tpu_id)
    BMimg = decoder.read(handle)  # here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    paddingatt = sail.PaddingAtrr()
    paddingatt.set_stx(0)
    paddingatt.set_sty(0)
    paddingatt.set_w(640)
    paddingatt.set_h(640)
    paddingatt.set_r(114)
    paddingatt.set_g(114)
    paddingatt.set_b(114)
    BMimg4 = bmcv.vpp_resize_padding(BMimg,640,640,paddingatt)
```

5.19.12 warp

对图像进行仿射变换。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: 智能视觉深度学习处理器

接口形式 1:

```
def warp(self,
    input: sail.BMImage | sail.BMImageArray,
    matrix: list,
    use_bilinear: int = 0,
    similar_to_opencv: bool = False)->sail.BMImage | sail.BMImageArray
```

参数说明 1:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- matrix: 2d list

2x3 的仿射变换矩阵。

- use_bilinear: int

是否使用双线性插值，默认为 0 使用最近邻插值，1 为双线性插值

- similar_to_opencv: bool

是否使用与 opencv 仿射变换对齐的接口

返回值说明 1:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    rotated_matrix = [[0.9996914396,-0.02484,0],[0.02484,0.9996914396,0]]
    BMimg6 = bmcv.warp(BMimg, rotated_matrix)
```

接口形式 2:

```
def warp(self,
    input: sail.BMImage | sail.BMImageArray,
    output: sail.BMImage | sail.BMImageArray,
    matrix: list,
    use_bilinear: int = 0,
    similar_to_opencv: bool = False)->int
```

参数说明 2:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- output：sail.BMImage | sail.BMImageArray

待返回的输出图像或图像数组。

- matrix: 2d list

2x3 的仿射变换矩阵。

- use_bilinear: int

是否使用双线性插值，默认为 0 使用最近邻插值，1 为双线性插值

- similar_to_opencv: bool

是否使用与 opencv 仿射变换对齐的接口

返回值说明 2:

如果成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    rotated_matrix = [[0.9996914396,-0.02484,0],[0.02484,0.9996914396,0]]
    output = sail.BMImage()
    ret = bmcv.warp(BMimg, output,rotated_matrix)
```

5.19.13 convert_to

对图像进行线性变换。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP+ 智能视觉深度学习处理器

接口形式 1:

```
def convert_to(self,
    input: sail.BMImage | sail.BMImageArray,
    alpha_beta: tuple)->sail.BMImage | sail.BMImageArray
```

参数说明 1:

- input：sail.BMImage | sail.BMImageArray

待处理的图像或图像数组。

- alpha_beta: tuple

分别为三个通道线性变换的系数 ((a0, b0), (a1, b1), (a2, b2))。

返回值说明 1:

- output：sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    BMimg5 = bmcv.convert_to(BMimg, alpha_beta)
```

接口形式 2:

```
def convert_to(self,
        input: sail.BMImage | sail.BMImageArray,
        output: sail.BMImage | sail.BMImageArray,
        alpha_beta: tuple)->None
```

参数说明 2:

- input：sail.BMImage | sail.BMImageArray

输入参数。待处理的图像或图像数组。

- output：sail.BMImage | sail.BMImageArray

输出参数。返回处理后的图像或图像数组。

- alpha_beta: tuple

分别为三个通道线性变换的系数 ((a0, b0), (a1, b1), (a2, b2))。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    BMimg5 = sail.BMImage()
    bmcv.convert_to(BMimg, BMimg5,alpha_beta)
```

5.19.14 yuv2bgr

将图像的格式从 YUV 转换为 BGR。

实现硬件 * BM1684: 智能视觉深度学习处理器 +VPP * BM1684X: VPP

接口形式:

```
def yuv2bgr(input: sail.BMImage | sail.BMImageArray)
        -> sail.BMImage | sail.BMImageArray
```

参数说明:

- input : sail.BMImage | sail.BMImageArray

待转换的图像。

返回值说明:

- output : sail.BMImage | sail.BMImageArray

返回转换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg5 = bmcv.yuv2bgr(BMimg)
```

5.19.15 rectangle

在图像上画一个矩形框。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def rectangle(self,
        image: sail.BMImage,
        x0: int,
        y0: int,
        w: int,
        h: int,
        color: tuple,
        thickness: int=1)->int
```

参数说明:

- image：sail.BMImage

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

- thickness：int

矩形框线条的粗细。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.rectangle(BMimg, 20, 20, 600, 600,(0,0,255),2)
```

5.19.16 fillRectangle

在图像上画一个矩形框。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def fillRectangle(self,
        image: sail.BMImage,
        x0: int,
```

(下页继续)

(续上页)

```
        y0: int,
        w: int,
        h: int,
        color: tuple)->int
```

参数说明:

- image：sail.BMImage

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.fillRectangle(BMimg, 20, 20, 600, 600,(0,0,255))
```

5.19.17 imwrite

将图像保存在特定文件。

实现硬件 * BM1684: VPP+ 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def imwrite(self, file_name: str, image: sail.BMImage)->int
```

参数说明:

- file_name : str

文件的名称。

- output : sail.BMImage

需要保存的图像。

返回值说明:

- process_status : int

如果保存成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmcv.imwrite("{}_{}.jpg".format(BMimg.width(),BMimg.height()),BMimg)
```

5.19.18 get_handle

获取 Bmcv 中的设备句柄 Handle。

接口形式:

```
def get_handle(self)->sail.Handle
```

返回值说明:

- handle: sail.Handle

Bmcv 中的设备句柄 Handle。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    handle1 = bmcv.get_handle()
```

5.19.19 crop_and_resize_padding

对图像进行裁剪并 resize，然后 padding。

实现硬件 * BM1684: VPP+ 智能视觉深度学习处理器 * BM1684X: VPP+ 智能视觉深度学习处理器

接口形式:

```
def crop_and_resize_padding(self,
                input: sail.BMImage,
                crop_x0: int,
                crop_y0: int,
                crop_w: int,
                crop_h: int,
                resize_w: int,
                resize_h: int,
                padding: PaddingAtrr,
                resize_alg=sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST)
            ->sail.BMImage
```

参数说明:

- input：sail.BMImage

待处理的图像。

- crop_x0：int

裁剪窗口在 x 轴上的起始点。

- crop_y0：int

裁剪窗口在 y 轴上的起始点。

- crop_w：int

裁剪窗口的宽。

- crop_h：int

裁剪窗口的高。

- resize_w：int

图像 resize 的目标宽度。

- resize_h：int

图像 resize 的目标高度。

- padding：sail.PaddingAtrr

padding 的配置信息。

- resize_alg：bmcv_resize_algorithm

resize 采用的插值算法。

返回值说明:

- output：sail.BMImage

返回处理后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0 = 100
    crop_y0 = 100
    crop_w = 200
    crop_h = 200
    resize_w = 300
    resize_h = 300

    paddingatt = sail.PaddingAtrr()
    paddingatt.set_stx(0)
    paddingatt.set_sty(0)
    paddingatt.set_w(300)
    paddingatt.set_h(300)
    paddingatt.set_r(114)
    paddingatt.set_g(114)
    paddingatt.set_b(114)
    padded_BMimg = bmcv.crop_and_resize_padding(
        BMimg,
        crop_x0,
        crop_y0,
        crop_w,
        crop_h,
        resize_w,
```

(下页继续)

(续上页)

```
        resize_h,
        paddingatt,
        sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST
    )
```

5.19.20 rectangle_

在图像上画一个矩形框。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def rectangle_(self,
        image: sail.bm_image,
        x0: int,
        y0: int,
        w: int,
        h: int,
        color: tuple,
        thickness: int=1)->int
```

参数说明:

- image：sail.bm_image

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

- thickness：int

矩形框线条的粗细。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.rectangle_(BMimg.data(), 20, 20, 600, 600,(0,0,255),2)
```

5.19.21 fillRectangle_

在图像上画一个矩形。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def fillRectangle_(self,
        image: sail.bm_image,
        x0: int,
        y0: int,
        w: int,
        h: int,
        color: tuple)->int
```

参数说明:

- image：sail.bm_image

待画框的图像。

- x0：int

矩形框在 x 轴上的起点。

- y0：int

矩形框在 y 轴上的起点。

- w：int

矩形框的宽度。

- h：int

矩形框的高度。

- color：tuple

矩形框的颜色。

- thickness：int

矩形框线条的粗细。

返回值说明:

如果画框成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.fillRectangle_(BMimg.data(), 20, 20, 600, 600,(0,0,255))
```

5.19.22 imwrite_

将图像保存在指定的文件。

实现硬件 * BM1684: VPP+ 智能视觉深度学习处理器 * BM1684X: VPP

接口形式:

```
def imwrite_(self, file_name: str, image: sail.bm_image)->int
```

参数说明:

- file_name : str

文件的名称。

- image : sail.bm_image

需要保存的图像。

返回值说明:

- process_status : int

如果保存成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
```

(下页继续)

(续上页)

```
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.imwrite_("{}_{}.jpg".format(BMimg.width(),BMimg.height()),BMimg.data())
```

5.19.23 convert_format

将图像的格式转换为 output 中的格式，并拷贝到 output。

实现硬件 * BM1684: VPP+ 智能视觉深度学习处理器 * BM1684X: VPP

接口形式 1:

```
def convert_format(self, input: sail.BMImage, output: sail.BMImage)->None
```

参数说明 1:

- input：sail.BMImage

输入参数。待转换的图像。

- output：sail.BMImage

输出参数。将 input 中的图像转化为 output 的图像格式并拷贝到 output。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = sail.BMImage()
    bmcv.convert_format(BMimg,output)
```

接口形式 2:

将一张图像转换成目标格式。

```
def convert_format(self, input: sail.BMImage, image_format:sail.bm_image_format_
↪ext)->sail.BMImage
```

参数说明 2:

- input：sail.BMImage

待转换的图像。

- image_format：sail.bm_image_format_ext

转换的目标格式。

返回值说明 2:

- output：sail.BMImage

返回转换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = bmcv.convert_format(BMimg,sail.FORMAT_BGR_PLANAR)
```

5.19.24 vpp_convert_format

利用 VPP 硬件加速图片的格式转换。

实现硬件 * BM1684: VPP * BM1684X: VPP

接口形式 1:

```
def vpp_convert_format(self, input: sail.BMImage, output: sail.BMImage)->None
```

参数说明 1:

- input：sail.BMImage

输入参数。待转换的图像。

- output：sail.BMImage

输出参数。将 input 中的图像转化为 output 的图像格式并拷贝到 output。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = sail.BMImage()
    bmcv.vpp_convert_format(BMimg,output)
```

接口形式 2:

将一张图像转换成目标格式。

```
def vpp_convert_format(self, input: sail.BMImage, image_format:sail.bm_image_
↪format_ext)->sail.BMImage
```

参数说明 2:

- input：sail.BMImage

待转换的图像。

- image_format：sail.bm_image_format_ext

转换的目标格式。

返回值说明 2:

- output：sail.BMImage

返回转换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = bmcv.vpp_convert_format(BMimg,sail.FORMAT_BGR_PLANAR)
```

5.19.25 putText

在图像上添加 text。

实现硬件 * BM1684: 处理器 * BM1684X: 处理器

接口形式:

```
def putText(self,
        input: sail.BMImage,
        text: str,
        x: int,
        y: int,
        color: tuple,
        fontScale: int,
        thickness: int)->int
```

参数说明:

- input：sail.BMImage

待处理的图像。

- text: str

需要添加的文本。

- x: int

添加的起始点位置。

- y: int

添加的起始点位置。

- color：tuple

字体的颜色。

- fontScale: int

字号的大小。

- thickness：int

字体的粗细。

返回值说明:

- process_status：int

如果处理成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.putText(BMimg, "snow person" , 20, 20, [0,0,255], 1.4, 2)
```

5.19.26 putText_

实现硬件 * BM1684: 处理器 * BM1684X: 处理器

接口形式:

```
def putText_(self,
        input: sail.bm_image,
        text: str,
```

(下页继续)

(续上页)

```
        x: int,
        y: int,
        color: tuple,
        fontScale: int,
        thickness: int)->int
```

参数说明:

- input：sail.bm_image

待处理的图像。

- text: str

需要添加的文本。

- x: int

添加的起始点位置。

- y: int

添加的起始点位置。

- color：tuple

字体的颜色。

- fontScale: int

字号的大小。

- thickness：int

字体的粗细。

返回值说明:

- process_status：int

如果处理成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.putText_(BMimg.data(), "snow person" , 20, 20, [0,0,255], 1.4, 2)
```


5.19.27 image_add_weighted

将两张图像按不同的权重相加。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: 智能视觉深度学习处理器

接口形式 1:

```
def image_add_weighted(self,
            input0: sail.BMImage,
            alpha: float,
            input1: sail.BMImage,
            beta: float,
            gamma: float,
            output: sail.BMImage)->None
```

参数说明 1:

- input0 : sail.BMImage

输入参数。待处理的图像 0。

- alpha : float

输入参数。两张图像相加的权重 alpha

- input1 : sail.BMImage

输入参数。待处理的图像 1。

- beta : float

输入参数。两张图像相加的权重 beta

- gamma : float

输入参数。两张图像相加的权重 gamma

- output: BMImage

输出参数。相加后的图像 output = input1 * alpha + input2 * beta + gamma

示例代码 1:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
```

(下页继续)

(续上页)

```
    bmg = sail.BMImage()
    bmcv.image_add_weighted(BMimg1,0.5,BMimg2,0.5,0.5,bmg)
```

接口形式 2:

```
def image_add_weighted(self,
            input0: sail.BMImage,
            alpha: float,
            input1: sail.BMImage,
            beta: float,
            gamma: float)->sail.BMImage
```

参数说明 2:

- input0 : sail.BMImage

输入参数。待处理的图像 0。

- alpha : float

输入参数。两张图像相加的权重 alpha

- input1 : sail.BMImage

输入参数。待处理的图像 1。

- beta : float

输入参数。两张图像相加的权重 beta

- gamma : float

输入参数。两张图像相加的权重 gamma

返回值说明 2:

- output: sail.BMImage

返回相加后的图像 output = input1 * alpha + input2 * beta + gamma

示例代码 2:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmg = bmcv.image_add_weighted(BMimg1,0.5,BMimg2,0.5,0.5)
```

5.19.28 image_copy_to

进行图像间的数据拷贝

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP+ 智能视觉深度学习处理器

接口形式:

```
def image_copy_to(self,
          input: BMImage|BMImageArray,
          output: BMImage|BMImageArray,
          start_x: int,
          start_y: int)->None
```

参数说明:

- input: BMImage|BMImageArray

输入参数。待拷贝的 BMImage 或 BMImageArray。

- output: BMImage|BMImageArray

输出参数。拷贝后的 BMImage 或 BMImageArray

- start_x: int

输入参数。拷贝到目标图像的起始点。

- start_y: int

输入参数。拷贝到目标图像的起始点。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmcv.image_copy_to(BMimg1,BMimg2,0,0)
```

5.19.29 image_copy_to_padding

进行 input 和 output 间的图像数据拷贝并 padding。

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: VPP+ 智能视觉深度学习处理器

接口形式:

```
def image_copy_to_padding(self,
            input: BMImage|BMImageArray,
            output: BMImage|BMImageArray,
            padding_r: int,
            padding_g: int,
            padding_b: int,
            start_x: int,
            start_y: int)->None
```

参数说明:

- input: BMImage|BMImageArray

输入参数。待拷贝的 BMImage 或 BMImageArray。

- output: BMImage|BMImageArray

输出参数。拷贝后的 BMImage 或 BMImageArray

- padding_r: int

输入参数。R 通道的 padding 值。

- padding_g: int

输入参数。G 通道的 padding 值。

- padding_b: int

输入参数。B 通道的 padding 值。

- start_x: int

输入参数。拷贝到目标图像的起始点。

- start_y: int

输入参数。拷贝到目标图像的起始点。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
```

(下页继续)

(续上页)

```
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmcv.image_copy_to_padding(BMimg1,BMimg2,128,128,128,0,0)
```

5.19.30 nms

利用智能视觉深度学习处理器进行 NMS

实现硬件 * BM1684: 智能视觉深度学习处理器 * BM1684X: 智能视觉深度学习处理器

接口形式:

```
def nms(self, input: numpy.ndarray, threshold: float)->numpy.ndarray
```

参数说明:

- input: numpy.ndarray

待处理的检测框的数组，shape 必须是（n,5）n<56000 [left,top,right,bottom,score]。

- threshold: float

nms 的阈值。

返回值说明:

- result: numpy.ndarray

返回 NMS 后的检测框数组。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    bmcv = sail.Bmcv(handle)
    input_boxes = np.array([
    [50, 50, 100, 100, 0.9],
    [60, 60, 110, 110, 0.85],
    [200, 200, 250, 250, 0.7],
    [130, 50, 180, 100, 0.8],
    [205, 205, 255, 255, 0.75]
    ])
    nms_threshold = 0.5
    selected_boxes  = bmcv.nms(input_boxes, nms_threshold)
    print(selected_boxes)
```

5.19.31 drawPoint

在图像上画点。

实现硬件 * BM1684: 处理器 * BM1684X: VPP

接口形式:

```
def drawPoint(self,
        image: BMImage,
        center: Tuple[int, int],
        color: Tuple[int, int, int],
        radius: int) -> int:
```

参数说明:

- image: BMImage

输入图像，在该 BMImage 上直接画点作为输出。

- center: Tuple[int, int]

点的中心坐标。

- color: Tuple[int, int, int]

点的颜色。

- radius: int

点的半径。

返回值说明

如果画点成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.drawPoint(BMimg, (320, 320), (0,255,255),2)
```

5.19.32 drawPoint_

在图像上画点。

实现硬件 * BM1684: 处理器 * BM1684X: VPP

接口形式:

```
def drawPoint_(self,
        image: bm_image,
        center: Tuple[int, int],
        color: Tuple[int, int, int],
        radius: int) -> int:
```

参数说明:

- image: bm_image

输入图像，在该 BMImage 上直接画点作为输出。

- center: Tuple[int, int]

点的中心坐标。

- color: Tuple[int, int, int]

点的颜色。

- radius: int

点的半径。

返回值说明

如果画点成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.drawPoint_(BMimg.data(), (320, 320), (0,255,255),2)
```

5.19.33 warp_perspective

对图像进行透视变换。

接口形式:

```
def warp_perspective(self,
          input: BMImage,
          coordinate: tuple,
          output_width: int,
          output_height: int,
          format: bm_image_format_ext = FORMAT_BGR_PLANAR,
          dtype: bm_image_data_format_ext = DATA_TYPE_EXT_1N_BYTE,
          use_bilinear: int = 0 ) -> BMImage:
```

参数说明:

- input: BMImage

待处理的图像。

- coordinate: tuple

变换区域的四顶点原始坐标。tuple(tuple(int,int))

例如 ((left_top.x, left_top.y), (right_top.x, right_top.y), (left_bottom.x, left_bottom.y), (right_bottom.x, right_bottom.y))

- output_width: Output width

输出图像的宽。

- output_height: Output height

输出图像的高。

- bm_image_format_ext: sail.Format

输出图像的格式。

- dtype: sail.ImgDtype

输出图像的数据类型。

- use_bilinear: bool

是否使用双线性插值。

返回值说明:

- output: image

输出变换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    img = bmcv.warp_perspective(BMimg, ((100, 100), (540, 100), (100, 380), (540, 380)),640,
↪640)
```

5.19.34 get_bm_data_type

将 ImgDtype 转换为 Dtype

接口形式:

```
def get_bm_data_type((self, format: sail.ImgDtype) -> sail.Dtype
```

参数说明:

- format: sail.ImgDtype

需要转换的类型。

返回值说明:

- ret: sail.Dtype

转换后的类型。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    type = bmcv.get_bm_data_type(sail.DATA_TYPE_EXT_FLOAT32)
```

5.19.35 get_bm_image_data_format

将 Dtype 转换为 ImgDtype。

接口形式:

```
def get_bm_image_data_format(self, dtype: sail.Dtype) -> sail.ImgDtype
```

参数说明:

- dtype: sail.Dtype

需要转换的 sail.Dtype

返回值说明:

- ret: sail.ImgDtype

返回转换后的类型。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    type = bmcv.get_bm_image_data_format(sail.BM_FLOAT32)
```

5.19.36 imdecode

从内存中载入图像到 BMImage 中。

接口形式:

```
def imdecode(self, data_bytes: bytes) -> sail.BMImage:
```

参数说明:

- data_bytes: bytes

系统内存中图像的 bytes

返回值说明:

- ret: sail.BMImage

返回解码后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    with open(image_name, 'rb') as image_file:
        image_data_bytes = image_file.read()
    bmcv = sail.Bmcv(handle)
    src_img = bmcv.imdecode(image_data_bytes)
```

5.19.37 imencode

编码一张图片，并返回编码后的数据。

接口形式:

```
def imencode(self, ext: str, img: BMImage) -> numpy.ndarray:
```

参数说明:

- ext: str

输入参数。图片编码格式。".jpg" , ".png" 等。

- img: BMImage

输入参数。输入图片，只支持 FORMAT_BGR_PACKED, DATA_TYPE_EXT_1N_BYTE 的图片。

返回值说明:

- ret: numpy.array

编码后放在系统内存中的数据。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.imencode(".jpg",BMimg)
```

5.19.38 fft

实现对 Tensor 的快速傅里叶变换。

接口形式:

```
fft(self, forward: bool, input_real: Tensor) -> list[Tensor]

fft(self, forward: bool, input_real: Tensor, input_imag: Tensor) -> list[Tensor]
```

参数说明:

- forward: bool

是否进行正向迁移。

- input_real: Tensor

输入的实数部分。

- input_imag: Tensor

输入的虚数部分。

返回值说明:

- ret: list[Tensor]

返回输出的实数部分和虚数部分。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    random_array1 = np.random.rand(1, 1, 512, 512).astype('float32'),
    random_array2 = np.random.rand(1, 1, 512, 512).astype('float32'),
    bmcv = sail.Bmcv(handle)
    input_real  = sail.Tensor(handle,random_array1,True)
    input_imag  = sail.Tensor(handle,random_array2,True)
    forward = True
    result_complex = bmcv.fft(forward,input_real,input_imag)
```

5.19.39 convert_yuv420p_to_gray

将 YUV420P 格式的图片转为灰度图。

接口形式 1:

```
def convert_yuv420p_to_gray(self, input: sail.BMImage, output: sail.BMImage)->None
```

参数说明 1:

- input：sail.BMImage

输入参数。待转换的图像。

- output：sail.BMImage

输出参数。转换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg2 = sail.BMImage()
    bmcv.convert_yuv420p_to_gray(BMimg1,BMimg2)
```

接口形式 2:

将 YUV420P 格式的图片转为灰度图。

```
def convert_yuv420p_to_gray_(self, input: sail.bm_image, output: sail.bm_image)->
↪None
```

参数说明 2:

- input：sail.bm_image

待转换的图像。

- output：sail.bm_image

转换后的图像。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmg = sail.BMImage()
    bmcv.convert_yuv420p_to_gray_(BMimg1.data(),bmg.data())
```

5.19.40 mat_to_bm_image

将 opencv 的 mat 转为 sail 的 BMImage。

接口形式 1:

```
def mat_to_bm_image(self, mat: numpy.ndarray[numpy.uint8]) -> BMImage:
```

参数说明 1:

- mat : numpy

输入参数。待转换的 opencv mat。

返回值说明:

- ret: sail.BMImage

返回转换后的 sail.BMImage。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    bmcv = sail.Bmcv(handle)
    opencv_mat = cv2.imread(image_name)
    sail_bm_image = bmcv.mat_to_bm_image(opencv_mat)
```

接口形式 2:

```
def mat_to_bm_image(self, mat: numpy.ndarray[numpy.uint8], img: BMImage) -> int:
```

参数说明 2:

- mat : numpy

待转换的 opencv mat。

- img : sail.BMImage

转换后的 BMImage。

返回值说明:

- ret: int

成功后返回 0

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    bmcv = sail.Bmcv(handle)
    opencv_mat = cv2.imread(image_name)
    BMimg2 = sail.BMImage()
    ret = bmcv.mat_to_bm_image(opencv_mat,BMimg2)
```

5.19.41 watermark_superpose

实现对图片添加多个水印。

接口形式:

```
def watermark_superpose(self,
image: sail.BMImage,
water_name:string,
bitmap_type: int,
pitch: int,
rects: list[list[int]],
color: tuple
        )->int
```

参数说明:

- image: sail.BMImage

输入图片

- water_name:string

水印文件路径

- bitmap_type: int

输入参数。水印类型, 值 0 表示水印为 8bit 数据类型 (有透明度信息), 值 1 表示水印为 1bit 数据类型 (无透明度信息)。

- pitch: int

输入参数。水印文件每行的 byte 数, 可理解为水印的宽。

- rects: list[list[int]]

输入参数。水印位置，包含每个水印起始点和宽高。

- color: tuple

输入参数。水印的颜色。

返回值说明:

- ret: int

返回是否成功

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path1"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmg = sail.BMImage()
    water_name = 'your_watermark_path'
    ret = bmcv.watermark_superpose(BMimg1,water_name,0,117,[[0,0,117,79],[0,90,117,79]],
↪[128,128,128])
```

5.19.42 polylines

可以实现在一张图像上画一条或多条线段，从而可以实现画多边形的功能，并支持指定线的颜色和线的宽度。

接口形式:

```
def polylines(self, image: BMImage, pts: list[list[tuple(int, int)]], isClosed: bool, color:␣
↪tuple(int, int, int), thickness: int = 1, shift: int = 0) -> int:
```

参数说明:

- img：sail.BMImage

输入图片。

- pts：list[list[tuple(int, int)]]

线段的起始点和终点坐标，可输入多个坐标点。图像左上角为原点，向右延伸为 x 方向，向下延伸为 y 方向。

- isClosed：bool

图形是否闭合。

- color：tuple(int, int, int)

画线的颜色，分别为 RGB 三个通道的值。

- thickness：int

画线的宽度，对于 YUV 格式的图像建议设置为偶数。

- shift：int

多边形缩放倍数，默认不缩放。缩放倍数为 (1/2)^shift。

返回值说明:

- ret: int

成功后返回 0

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bm = bmcv.vpp_convert_format(BMimg1,sail.FORMAT_YUV444P)
    ret = bmcv.polylines(bm,[[(10,20),(40,80)]],True,[128,128,128])
```

5.19.43 mosaic

该接口用于在图像上打一个或多个马赛克。

接口形式:

```
def mosaic(self, mosaic_num: int, img: sail.BMImage, rects: list[list[int,int,int,int]], is_
↪expand:int) -> int
```

参数说明:

- mosaic_num : int

马赛克数量，指 rects 中列表长度。

- img : sail.BMImage

待转换的图像。

- rects : list[list[int,int,int,int]]

多个马赛克位置，列表中每个元素中参数为 [马赛克在 x 轴起始点, 马赛克在 y 轴起始点, 马赛克宽, 马赛克高]

- is_expand : int

是否扩列。值为 0 时表示不扩列, 值为 1 时表示在原马赛克周围扩列一个宏块 (8 个像素)。

返回值说明:

- ret: int

成功后返回 0

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.mosaic(2, BMimg1, [[10,10,100,2000],[500,500,1000,100]], 1)
```

5.19.44 gaussian_blur

该接口用于对图像进行高斯滤波。**注意：旧版本 SDK 并不支持 BM1684X，当前 SDK 是否支持请查询《BMCV 开发参考手册》，BMCV API 页面查看。**

接口形式:

```
def gaussian_blur(self, input: BMImage, kw: int, kh : int, sigmaX : float, sigmaY : float = 0.
↪0) -> BMImage:
```

参数说明:

- input：sail.BMImage

待转换的图像。

- kw：int

kernel 在 width 方向上的大小。

- kh：int

kernel 在 height 方向上的大小。

- sigmaX：float

X 方向上的高斯核标准差。

- sigmaY：float

Y 方向上的高斯核标准差。如果为 0 则表示与 X 方向上的高斯核标准差相同。默认为 0。

返回值说明:

- output：sail.BMImage

返回经过高斯滤波的图像。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    handle = sail.Handle(0)
    bmcv = sail.Bmcv(handle)


    bmimg = sail.BMImage()
    decoder = sail.Decoder("your_img.jpg",True,0)
    bmimg = decoder.read(handle)

    print(bmimg.format())
    output = bmcv.gaussian_blur(bmimg, 3, 3, 0.1)

    bmcv.imwrite("out.jpg",output)
```

5.19.45 transpose

该接口可以实现图片宽和高的转置。

接口形式 1:

```
def transpose(self, src: sail.BMImage) -> sail.BMImage:
```

参数说明 1:

- src：sail.BMImage

待转换的图像。

返回值说明 1:

- output: sail.BMImage:

返回转换后的图像。

接口形式 2:

```
def transpose(self, src: sail.BMImage, dst: sail.BMImage) -> int:
```

参数说明 2:

- src：sail.BMImage

待转换的图像。

- dst：sail.BMImage

输出图像的 sail.BMImage 结构体。

返回值说明 2:

- ret：int

成功返回 0，否则返回非 0 值。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':

    handle = sail.Handle(0)
    bmcv = sail.Bmcv(handle)
    bmimg = sail.BMImage()
    decoder = sail.Decoder("your_img.jpg",True,0)
    bmimg = decoder.read(handle)
    img = bmcv.convert_format(bmimg,sail.Format.FORMAT_GRAY)
    print("readed")
    print(img.format())
    output = bmcv.transpose(img)

    bmcv.imwrite("out.jpg",output)
```

5.19.46 Sobel

边缘检测 Sobel 算子。

注意：请查询《BMCV 开发参考手册/BMCV API》确认当前算子是否适配 BM1684X。

接口形式 1:

```
def Sobel(self, input: BMImage, output: BMImage, dx: int, dy: int, ksize: int = 3, scale: float
↪= 1, delta: float = 0) -> int:
```

参数说明 1:

- input：sail.BMImage

待转换的图像。

- output：sail.BMImage

转换后的图像。

- dx：int

x 方向上的差分阶数。

- dy：int

y 方向上的差分阶数。

- ksize：int

Sobel 核的大小，必须是-1,1,3,5 或 7。其中特殊的，如果是-1 则使用 3×3 Scharr 滤波器，如果是 1 则使用 3×1 或者 1×3 的核。默认值为 3。

- scale：float

对求出的差分结果乘以该系数，默认值为 1。

- delta：float

在输出最终结果之前加上该偏移量，默认值为 0。

返回值说明 1:

- ret: int

成功后返回 0

接口形式 2:

```
def Sobel(self, input: BMImage, dx: int, dy: int, ksize: int = 3, scale: float = 1, delta: float =␣
↪0) -> BMImage:
```

参数说明 2:

- input：sail.BMImage

待转换的图像。

- dx：int

x 方向上的差分阶数。

- dy：int

y 方向上的差分阶数。

- ksize：int

Sobel 核的大小，必须是-1,1,3,5 或 7。其中特殊的，如果是-1 则使用 3×3 Scharr 滤波器，如果是 1 则使用 3×1 或者 1×3 的核。默认值为 3。

- scale：float

对求出的差分结果乘以该系数，默认值为 1。

- delta：float

在输出最终结果之前加上该偏移量，默认值为 0。

返回值说明 2:

- output: sail.BMImage

返回转换后的图像。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    handle = sail.Handle(0)
    bmcv = sail.Bmcv(handle)

    bmimg = sail.BMImage()
    decoder = sail.Decoder("your_img.jpg",True,1)
    bmimg = decoder.read(handle)

    print(bmimg.format())
```

（下页继续）

(续上页)

```
output = bmcv.Sobel(bmimg, 1, 1)

bmcv.imwrite("out.jpg",output)
```

5.20 sail.MultiDecoder

多路解码接口，支持同时解码多路视频。

5.20.1 __init__

接口形式:

```
def __init__(self,
             queue_size: int = 10,
             tpu_id: int = 0,
             discard_mode: int = 0)
```

参数说明:

- queue_size: int

输入参数。每路视频，解码缓存图像队列的长度。

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id，默认为 0。

- discard_mode: int

输入参数。缓存达到最大值之后，数据的丢弃策略。0 表示不再放数据进缓存；1 表示先从队列中取出队列头的图片，丢弃之后再将解码出来的图片缓存进去。默认为 0。

5.20.2 set_read_timeout

设置读取图片的超时时间，对 read 和 read_ 接口生效，超时之后仍然没有获取到图像，结果就会返回。

接口形式:

```
def set_read_timeout(self, timeout: int) -> None
```

参数说明:

- timeout: int

输入参数。超时时间，单位是秒。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    multiDecoder.set_read_timeout(100)
```

5.20.3 add_channel

添加一个通道。

接口形式:

```
def add_channel(self,
                file_path: str,
                frame_skip_num: int = 0) -> int
```

参数说明:

- file_path: str

输入参数。视频的路径或者链接。

- frame_skip_num: int

输入参数。解码缓存的主动丢帧数，默认是 0，不主动丢帧。

返回值说明

返回视频对应的唯一的通道号。类型为整形。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
```

5.20.4 del_channel

删除一个已经添加的视频通道。

接口形式:

```
def del_channel(self, channel_idx: int) -> int
```

参数说明:

- channel_idx: int

输入参数。将要删除视频的通道号。

返回值说明

成功返回 0，其他值时表示失败。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []

    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
    ret = multiDecoder.del_channel(0)
    if(ret!=0):
        print("delete channel error!")
```

5.20.5 clear_queue

清除指定通道的图片缓存。

接口形式:

```
def clear_queue(self, channel_idx: int) -> int
```

参数说明:

- channel_idx: int

输入参数。将要删除视频的通道号。

返回值说明:

成功返回 0，其他值时表示失败。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
    ret = multiDecoder.clear_queue(0)
    if(ret!=0):
        print(" Clear failure!")
```

5.20.6 read

从指定的视频通道中获取一张图片。

接口形式 1:

```
def read(self,
        channel_idx: int,
        image: BMImage,
        read_mode: int = 0) -> int
```

参数说明 1:

- channel_idx: int

输入参数。指定的视频通道号。

- image: BMImage

输出参数。解码出来的图片。

- read_mode: int

输入参数。获取图片的模式，0 表示不等待，直接从缓存中读取一张，无论有没有读取到都会返回。其他的表示等到获取到图片之后或等待时间超时再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            bmimg = sail.BMImage()
            ret = multiDecoder.read(idx,bmimg,1)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

接口形式 2:

```
def read(self, channel_idx: int) -> BMImage
```

参数说明 2:

- channel_idx: int

输入参数。指定的视频通道号。

返回值说明 2:

返回解码出来的图片，类型为 BMImage。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
```

（下页继续）

(续上页)

```
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            bmimg = multiDecoder.read(idx)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

5.20.7 read_

从指定的视频通道中获取一张图片，通常是要和 BMImageArray 一起使用。

接口形式 1:

```
def read_(self,
        channel_idx: int,
        image: bm_image,
        read_mode: int=0) -> int
```

参数说明 1:

- channel_idx: int

输入参数。指定的视频通道号。

- image: bm_image

输出参数。解码出来的图片。

- read_mode: int

输入参数。获取图片的模式，0 表示不等待，直接从缓存中读取一张，无论有没有读取到都会返回。其他的表示等到获取到图片之后或等待时间超时再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

示例代码:

```
import sophon.sail as sail
if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
```

(下页继续)

(续上页)

```
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            img = sail.BMImage()
            bmimg = img.data()
            ret = multiDecoder.read_(idx,bmimg,1)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

接口形式 2:

```
def read_(self, channel_idx: int) -> int:
    """ Read a bm_image from the MultiDecoder with a given channel.
```

参数说明 2:

- channel_idx: int

输入参数。指定的视频通道号。

返回值说明 2:

返回解码出来的图片，类型为 bm_image。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
```

(下页继续)

(续上页)

```
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            bmimg = multiDecoder.read_(idx)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

5.20.8 reconnect

重连相应的通道的视频。

接口形式:

```
def reconnect(self, channel_idx: int) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

返回值说明

成功返回 0，其他值时表示失败。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
    ret = multiDecoder.reconnect(0)
    print(ret)
```

5.20.9 get_frame_shape

获取相应通道的图像 shape。

接口形式:

```
def get_frame_shape(self, channel_idx: int) -> list[int]
```

参数说明:

输入参数。输入图像的通道号。

返回值说明

返回一个由 1，通道数，图像高度，图像宽度组成的 list。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        print(multiDecoder.get_frame_shape(idx))
```

5.20.10 set_local_flag

设置视频是否为本地视频。如果不调用则表示为视频为网络视频流。

接口形式:

```
def set_local_flag(self, flag: bool) -> None:
```

参数说明:

- flag: bool

标准位，如果为 True，每路视频每秒固定解码 25 帧

示例代码:

```
import sophon.sail as sail
```

(下页继续)

(续上页)

```
if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    multiDecoder.set_local_flag(True)
```

5.20.11 get_channel_fps

获取指定视频通道的视频帧数

接口形式:

```
def get_channel_fps(self, channel_idx: int) -> float:
```

参数说明:

- channel_idx: int

指定需要获取视频帧数的视频通道号

返回值说明

返回指定视频通道的视频帧数

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        print(multiDecoder.get_channel_fps(idx))
```

get_drop_num

获取丢帧数。

接口形式:

```
def get_drop_num(self, channel_idx: int) -> int:
```

参数说明:

输入参数。输入图像的通道号。

返回值说明

返回一个数代表丢帧数

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        print(multiDecoder.get_drop_num(idx))
```

5.20.12 reset_drop_num

设置丢帧数为 0。

接口形式:

```
def reset_drop_num(self, channel_idx: int) -> None:
```

参数说明:

输入参数。输入图像的通道号。

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
```

(下页继续)

(续上页)

```
        exit(-1)
    channel_list.append(idx)
    multiDecoder.reset_drop_num(idx)
```

5.21 sail.sail_resize_type

图像预处理对应的预处理方法。

接口形式:

```
sail_resize_type.BM_RESIZE_VPP_NEAREST
sail_resize_type.BM_RESIZE_TPU_NEAREST
sail_resize_type.BM_RESIZE_TPU_LINEAR
sail_resize_type.BM_RESIZE_TPU_BICUBIC
sail_resize_type.BM_PADDING_VPP_NEAREST
sail_resize_type.BM_PADDING_TPU_NEAREST
sail_resize_type.BM_PADDING_TPU_LINEAR
sail_resize_type.BM_PADDING_TPU_BICUBIC
```

参数说明:

- BM_RESIZE_VPP_NEAREST

使用 VPP，最近邻的方法进行图像尺度变换。

- BM_RESIZE_TPU_NEAREST

使用智能视觉深度学习处理器，最近邻的方法进行图像尺度变换。

- BM_RESIZE_TPU_LINEAR

使用智能视觉深度学习处理器，线性插值的方法进行图像尺度变换。

- BM_RESIZE_TPU_BICUBIC

使用智能视觉深度学习处理器，双三次插值的方法进行图像尺度变换。

- BM_PADDING_VPP_NEAREST

使用 VPP，最近邻的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_NEAREST

使用智能视觉深度学习处理器，最近邻的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_LINEAR

使用智能视觉深度学习处理器，线性插值的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_BICUBIC

使用智能视觉深度学习处理器，双三次插值的方法进行带 padding 的图像尺度变换。

5.22 sail.ImagePreProcess

通用预处理接口，内部使用线程池的方式实现。

5.22.1 __init__

接口形式:

```
def __init__(self,
             batch_size: int,
             resize_mode:sail_resize_type,
             tpu_id: int=0,
             queue_in_size: int=20,
             queue_out_size: int=20,
             use_mat_output: bool = False)
```

参数说明:

- batch_size: int

输入参数。输出结果的 batch size。

- resize_mode: sail_resize_type

输入参数。内部尺度变换的方法。

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id，默认为 0。

- queue_in_size: int

输入参数。输入图像队列缓存的最大长度，默认为 20。

- queue_out_size: int

输入参数。输出 Tensor 队列缓存的最大长度，默认为 20。

- use_mat_output: bool

输入参数。是否使用 OpenCV 的 Mat 作为图片的输出，默认为 False，不使用。

5.22.2 SetResizeImageAtrr

设置图像尺度变换的属性。

接口形式:

```
def SetResizeImageAtrr(self,
             output_width: int,
             output_height: int,
             bgr2rgb: bool,
             dtype: ImgDtype) -> None
```

参数说明:

· output_width: int

输入参数。尺度变换之后的图像宽度。

· output_height: int

输入参数。尺度变换之后的图像高度。

· bgr2rgb: bool

输入参数。是否将图像有 BGR 转换为 GRB。

· dtype: ImgDtype

输入参数。图像尺度变换之后的数据类型，当前版本只支持 BM_FLOAT32,BM_INT8,BM_UINT8。可根据模型的输入数据类型设置。

5.22.3 SetPaddingAtrr

设置 Padding 的属性，只有在 resize_mode 为 BM_PADDING_VPP_NEAREST、BM_PADDING_TPU_NEAREST、BM_PADDING_TPU_LINEAR、BM_PADDING_TPU_BICUBIC 时生效。

接口形式:

```
def SetPaddingAtrr(self,
          padding_b: int=114,
          padding_g: int=114,
          padding_r: int=114,
          align: int=0) -> None
```

参数说明: * padding_b: int

输入参数。要 pdding 的 b 通道像素值，默认为 114。

· padding_g: int

输入参数。要 pdding 的 g 通道像素值，默认为 114。

· padding_r: int

输入参数。要 pdding 的 r 通道像素值，默认为 114。

· align: int

输入参数。图像填充为位置，0 表示从左上角开始填充，1 表示居中填充，默认为 0。

5.22.4 SetConvertAtrr

设置线性变换的属性。

接口形式:

```
def SetConvertAtrr(self, alpha_beta) -> int
```

参数说明:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。输入参数。

  a0 描述了第 0 个 channel 进行线性变换的系数;

  b0 描述了第 0 个 channel 进行线性变换的偏移;

  a1 描述了第 1 个 channel 进行线性变换的系数;

  b1 描述了第 1 个 channel 进行线性变换的偏移;

  a2 描述了第 2 个 channel 进行线性变换的系数;

  b2 描述了第 2 个 channel 进行线性变换的偏移;

返回值说明:

设置成功返回 0，其他值时设置失败。

5.22.5 PushImage

送入数据。

接口形式:

```
def PushImage(self,
    channel_idx: int,
    image_idx: int,
    image: BMImage) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- image: BMImage

输入参数。输入图像。

返回值说明:

设置成功返回 0，其他值时表示失败。

5.22.6 GetBatchData

获取处理的结果。

接口形式:

```
def GetBatchData(self)
   -> tuple[Tensor, list[BMImage],list[int],list[int],list[list[int]]]
""" Get the Batch Data object
```

返回值说明: tuple[data, images, channels, image_idxs, padding_attrs]

- data: Tensor

  处理后的结果 Tensor。

- images: list[BMImage]

  原始图像序列。

- channels: list[int]

  原始图像的通道序列。

- image_idxs: list[int]

  原始图像的编号序列。

- padding_attrs: list[list[int]]

  填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度

5.22.7 set_print_flag

设置打印日志的标志位，不调用此接口时不打印日志。

接口形式:

```
def set_print_flag(self, flag: bool) -> None:
```

返回值说明:

- flag: bool

打印的标志位，False 时表示不打印，True 时表示打印。

示例代码:

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

if __name__ == '__main__':
    tpu_id = 0
```

(下页继续)

(续上页)

```
    batch_size = 1
    image_path = "./data/zidane.jpg"
    handle = sail.Handle(tpu_id)

    alpha_beta = (1, 0), (1, 0), (1, 0)
    decoder = sail.Decoder(image_path, False, tpu_id)

    sail_ipp = sail.ImagePreProcess(batch_size, sail.sail_resize_type.BM_RESIZE_VPP_
↪NEAREST, tpu_id, 20, 20, False)

    sail_ipp.SetResizeImageAtrr(640, 640, False, sail.ImgDtype.DATA_TYPE_EXT_1N_
↪BYTE)
    ret1 = sail_ipp.SetConvertAtrr(alpha_beta)
    # sail_ipp.set_print_flag(True)
    bm_i = sail.BMImage()
    for i in range(0, batch_size):
        decoder.read(handle, bm_i)
        sail_ipp.PushImage(0, i, bm_i)

    result = sail_ipp.GetBatchData()
    decoder.release()

    tensor = result[0]
    t_npy = tensor.asnumpy()
    result_img = t_npy[0].transpose(1, 2, 0)

    raw_img = cv.imread(image_path)
    resize_img = cv.resize(raw_img, (640, 640), interpolation=cv.INTER_NEAREST)
    max_diff = abs((resize_img.astype(int) - result_img.astype(int)).max())
    min_diff = abs((resize_img.astype(int) - result_img.astype(int)).min())
    diff = max(max_diff, min_diff)
    print(max_diff,min_diff,diff)
```

5.23 sail.TensorPTRWithName

带有名称的 Tensor

5.23.1 get_name

获取 Tensor 的名称

接口形式:

```
def get_name(self) -> str
```

返回值说明:

返回 Tensor 的名称。

5.23.2 get_data

获取 Tensor

接口形式:

```
def get_data(self) -> Tensor
```

返回值说明:

返回 Tensor

5.24 sail.EngineImagePreProcess

带有预处理功能的图像推理接口，内部使用线程池的方式，Python 下面有更高的效率。

5.24.1 __init__

接口形式:

```
def __init__(self,
             bmodel_path: str,
             tpu_id: int,
             use_mat_output: bool = False,
             core_list:list = [])
```

参数说明: * bmodel_path: str

输入参数。输入模型的路径。

- tpu_id: int

输入参数。使用的智能视觉深度学习处理器 id。

- use_mat_output: bool

输入参数。是否使用 OpenCV 的 Mat 作为图片的输出，默认为 False，不使用。

- use_mat_output: bool

输入参数。使用支持多核推理的处理器和 bmodel 时，可以选择推理时使用的多个核心，默认使用从 core0 开始的 N 个 core 来做推理，N 由当前 bmodel 决定。对于仅支持单核推理的处理器和 bmodel 模型，仅支持选择推理使用的单个核心，参数的输入列表长度必须为 1，若传入列表长度大于 1，将自动在 0 号核心上推理。默认为空不指定时，将默认从 0 号核心开始的 N 个 core 来做推理。

5.24.2 InitImagePreProcess

初始化图像预处理模块。

接口形式:

```
def InitImagePreProcess(self,
            resize_mode: sail_resize_type,
            bgr2rgb: bool = False,
            queue_in_size: int = 20,
            queue_out_size: int = 20) -> int
```

参数说明:

- resize_mode: sail_resize_type

输入参数。内部尺度变换的方法。

- bgr2rgb: bool

输入参数。是否将图像有 BGR 转换为 GRB。

- queue_in_size: int

输入参数。输入图像队列缓存的最大长度，默认为 20。

- queue_out_size: int

输入参数。预处理结果 Tensor 队列缓存的最大长度，默认为 20。

返回值说明:

成功返回 0，其他值时失败。

5.24.3 SetPaddingAtrr

设置 Padding 的属性，只有在 resize_mode 为 BM_PADDING_VPP_NEAREST、BM_PADDING_TPU_NEAREST、BM_PADDING_TPU_LINEAR、BM_PADDING_TPU_BICUBIC 时生效。

接口形式:

```
def SetPaddingAtrr(self,
            padding_b:int=114,
            padding_g:int=114,
            padding_r:int=114,
            align:int=0) -> int
```

参数说明: * padding_b: int

输入参数。要 pdding 的 b 通道像素值，默认为 114。

- padding_g: int

输入参数。要 pdding 的 g 通道像素值，默认为 114。

- padding_r: int

输入参数。要 pdding 的 r 通道像素值，默认为 114。

- align: int

输入参数。图像填充为位置，0 表示从左上角开始填充，1 表示居中填充，默认为 0。

返回值说明:

成功返回 0，其他值时失败。

5.24.4 SetConvertAtrr

设置线性变换的属性。

接口形式:

```
def SetConvertAtrr(self, alpha_beta) -> int:
```

参数说明:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。输入参数。

  a0 描述了第 0 个 channel 进行线性变换的系数;

  b0 描述了第 0 个 channel 进行线性变换的偏移;

  a1 描述了第 1 个 channel 进行线性变换的系数;

  b1 描述了第 1 个 channel 进行线性变换的偏移;

  a2 描述了第 2 个 channel 进行线性变换的系数;

  b2 描述了第 2 个 channel 进行线性变换的偏移;

返回值说明:

设置成功返回 0，其他值时设置失败。

5.24.5 PushImage

送入图像数据

接口形式:

```
def PushImage(self,
              channel_idx: int,
              image_idx: int,
              image: BMImage) -> int
```

参数说明: * channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- image: BMImage

输入参数。输入的图像。

返回值说明:

成功返回 0，其他值时失败。

5.24.6 GetBatchData_Npy

获取一个 batch 的推理结果，调用此接口时，由于返回的结果类型为 BMImage，所以 use_mat_output 必须为 False。

接口形式:

```
def GetBatchData_Npy(self)
-> tuple[[dict[str, ndarray], list[BMImage],list[int],list[int],list[list[int]]]]
```

返回值说明:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: dict[str, ndarray]

推理结果。

- ost_images: list[BMImage]

原始图片序列。

- channels: list[int]

结果对应的原始图片的通道序列。

- image_idxs: list[int]

结果对应的原始图片的编号序列。

- padding_attrs: list[list[int]]

填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

5.24.7 GetBatchData_Npy2

获取一个 batch 的推理结果，调用此接口时，由于返回的结果类型为 numpy.ndarray[numpy.uint8]，所以 use_mat_output 必须为 True。

接口形式:

```
def GetBatchData_Npy2(self)
   -> tuple[dict[str, ndarray], list[numpy.ndarray[numpy.uint8]],list[int],list[int],list[list[int]]]
```

返回值说明:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: dict[str, ndarray]

推理结果。

- ost_images: list[numpy.ndarray[numpy.uint8]]

原始图片序列。

- channels: list[int]

结果对应的原始图片的通道序列。

- image_idxs: list[int]

结果对应的原始图片的编号序列。

- padding_attrs: list[list[int]]

填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

5.24.8 GetBatchData

获取一个 batch 的推理结果，调用此接口时，由于返回的结果类型为 BMImage，所以 use_mat_output 必须为 False。

接口形式:

```
def GetBatchData(self,
        need_d2s: bool = True)
        -> tuple[list[TensorPTRWithName], list[BMImage],list[int],list[int],list[list[int]]]
```

参数说明:

- need_d2s: bool

是否需要将数据搬运至系统内存，默认为 True, 需要搬运。

返回值说明:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: list[TensorPTRWithName]

推理结果。

- ost_images: list[BMImage]

原始图片序列。

- channels: list[int]

结果对应的原始图片的通道序列。

- image_idxs: list[int]

结果对应的原始图片的编号序列。

- padding_attrs: list[list[int]]

填充图像的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

5.24.9 get_graph_name

获取模型的运算图名称。

接口形式:

```
def get_graph_name(self) -> str
```

返回值说明:

返回模型的第一个运算图名称。

5.24.10 get_input_width

获取模型输入的宽度。

接口形式:

```
def get_input_width(self) -> int
```

返回值说明:

返回模型输入的宽度。

5.24.11 get_input_height

获取模型输入的高度。

接口形式:

```
def get_input_height(self) -> int
```

返回值说明:

返回模型输入的宽度。

5.24.12 get_output_names

获取模型输出 Tensor 的名称。

接口形式:

```
def get_output_names(self) -> list[str]
```

返回值说明:

返回模型所有输出 Tensor 的名称。

5.24.13 get_output_shape

获取指定输出 Tensor 的 shape

接口形式:

```
def get_output_shape(self, tensor_name: str) -> list[int]
```

参数说明:

- tensor_name: str

指定的输出 Tensor 的名称。

返回值说明:

返回指定输出 Tensor 的 shape。

示例代码:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    image_path = "./data/zidane.jpg"
    decoder = sail.Decoder(image_path, True, dev_id)
    bmodel_path = '../../../sophon-demo/sample/YOLOv5/models/BM1684X/yolov5s_v6.1_
↪3output_int8_1b.bmodel'
    alpha_beta = (1.0/255.0, 0), (1.0/255.0, 0), (1.0/255.0, 0)

    resize_type = sail.sail_resize_type.BM_PADDING_TPU_LINEAR
    sail_engineipp = sail.EngineImagePreProcess(bmodel_path, tpu_id, False)
    sail_engineipp.InitImagePreProcess(resize_type, False, 20, 20)

    sail_engineipp.SetPaddingAtrr()
    sail_engineipp.SetConvertAtrr(alpha_beta)


    get_i_w = sail_engineipp.get_input_width()
```

(下页继续)

(续上页)

```
get_i_h = sail_engineipp.get_input_height()
output_name = sail_engineipp.get_output_names()[0]
output_shape = sail_engineipp.get_output_shape(output_name)

bm_i = sail.BMImage()
decoder.read(handle, bm_i)
sail_engineipp.PushImage(0, 0, bm_i)

res = sail_engineipp.GetBatchData(True)
print(output_name,output_shape,get_i_h,get_i_w,res)
```

5.25 sail.algo_yolov5_post_1output

针对以单输出 YOLOv5 模型的后处理接口，内部使用线程池的方式实现。

5.25.1 __init__

接口形式:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

参数说明:

- shape: list[int]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

5.25.2 push_npy

输入数据，只支持 batchsize 为 1 的输入，或者输入之前将数据拆分之后再送入接口。

接口形式:

```
def push_npy(self,
        channel_idx: int,
        image_idx: int,
        data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
        dete_threshold: float,
        nms_threshold: float,
        ost_w: int,
        ost_h: int,
        padding_left: int,
        padding_top: int,
        padding_width: int,
        padding_height: int) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值。

- ost_w: int

输入参数。原始图片的宽。

- ost_h: int

输入参数。原始图片的高。

- padding_left: int

输入参数。填充图像的起始点坐标 x，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_top: int

输入参数。填充图像的起始点坐标 y，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_width: int

输入参数。填充图像的宽度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_height: int

输入参数。填充图像的高度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

返回值说明:

成功返回 0，其他值表示失败。

5.25.3 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像序列的通道号。

- image_idx: int

输入参数。输入图像序列的编号。

- input_data: list[TensorPTRWithName]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值序列。

- nms_threshold: float

输入参数。nms 阈值序列。

- ost_w: int

输入参数。原始图片序列的宽。

- ost_h: int

输入参数。原始图片序列的高。

- padding_attrs: list[list[int]]

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

5.25.4 get_result_npy

获取最终的检测结果

接口形式:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg"
    bmodel_name = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/yolov5s_v6.
↪1_1output_int8_1b.bmodel"
    decoder = sail.Decoder(image_name,True,tpu_id)
    bmimg = decoder.read(handle)
    engine_image_pre_process = sail.EngineImagePreProcess(bmodel_name, tpu_id, 0)
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_
↪TPU_LINEAR, True, 10, 10)
    engine_image_pre_process.SetPaddingAtrr(114,114,114,1)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    engine_image_pre_process.SetConvertAtrr(alpha_beta)
    ret = engine_image_pre_process.PushImage(0,0, bmimg)
    output_tensor_map, ost_images, channels ,imageidxs, paddding_attrs = engine_image_
↪pre_process.GetBatchData(True)
    width_list = []
    height_list= []
    for index, channel in enumerate(channels):
        width_list.append(ost_images[index].width())
        height_list.append(ost_images[index].height())
    yolov5_post = sail.algo_yolov5_post_1output([1, 25200, 85],640,640,10)
    dete_thresholds = np.ones(len(channels),dtype=np.float32)
    nms_thresholds = np.ones(len(channels),dtype=np.float32)
    dete_thresholds = 0.2*dete_thresholds
    nms_thresholds = 0.5*nms_thresholds
    ret = yolov5_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds,
↪ nms_thresholds, width_list, height_list, paddding_attrs)
    # 以下是利用push_npy接口推送 numpy 数据的示例
    .. for index, channel in enumerate(channels):
    ..     ret = yolov5_post.push_npy(channel, index, output_tensor_map[index].get_data().
↪asnumpy(), 0.2, 0.5,
    ..             ost_images[index].width(), ost_images[index].height(),
    ..             paddding_attrs[index][0], paddding_attrs[index][1], paddding_attrs[index][2],␣
↪paddding_attrs[index][3])
    objs, channel, image_idx = yolov5_post.get_result_npy()
    print(objs, channel, image_idx)
```

5.26 sail.algo_yolov5_post_3output

针对以三输出 YOLOv5 模型的后处理接口，内部使用线程池的方式实现。

5.26.1 __init__

接口形式:

```
def __init__(
        self,
        shape: list[list[int]],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

参数说明:

- shape: list[list[int]]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

5.26.2 push_data

输入数据，支持任意 batchsize 的输入。

接口形式:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
```

(下页继续)

(续上页)

```
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

参数说明:

- channel_idx: list[int]

输入参数。输入图像序列的通道号。

- image_idx: list[int]

输入参数。输入图像序列的编号。

- input_data: list[TensorPTRWithName],

输入参数。输入数据，包含三个输出。

- dete_threshold: list[float]

输入参数。检测阈值序列。

- nms_threshold: list[float]

输入参数。nms 阈值序列。

- ost_w: list[int]

输入参数。原始图片序列的宽。

- ost_h: list[int]

输入参数。原始图片序列的高。

- padding_attrs: list[list[int]]

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

5.26.3 get_result_npy

获取最终的检测结果

接口形式:

```
def get_result_npy(self)
        -> tuple[tuple[int, int, int, int, int, float],int, int]
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

5.26.4 reset_anchors

更新 anchor 尺寸.

接口形式:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

参数说明:

- anchors_new: list[list[list[int]]]

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
```

(下页继续)

(续上页)

```
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg"
    bmodel_name = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/yolov5s_v6.
↪1_3output_int8_1b.bmodel"
    decoder = sail.Decoder(image_name,True,tpu_id)
    bmimg = decoder.read(handle)
    engine_image_pre_process = sail.EngineImagePreProcess(bmodel_name, tpu_id, 0)
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_
↪TPU_LINEAR, True, 10, 10)
    engine_image_pre_process.SetPaddingAtrr(114,114,114,1)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    engine_image_pre_process.SetConvertAtrr(alpha_beta)
    ret = engine_image_pre_process.PushImage(0,0, bmimg)
    engine_image_pre_process
    output_tensor_map, ost_images, channels ,imageidxs, padding_atrr = engine_image_
↪pre_process.GetBatchData(True)
    width_list = []
    height_list= []
    for index, channel in enumerate(channels):
        width_list.append(ost_images[index].width())
        height_list.append(ost_images[index].height())
    yolov5_post = sail.algo_yolov5_post_3output([[1, 3, 20, 20, 85],[1, 3, 40, 40, 85],[1, 3, 80,
↪ 80, 85]],640,640,10)
    dete_thresholds = np.ones(len(channels),dtype=np.float32)
    nms_thresholds = np.ones(len(channels),dtype=np.float32)
    dete_thresholds = 0.2*dete_thresholds
    nms_thresholds = 0.5*nms_thresholds
    ret = yolov5_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds,
↪ nms_thresholds, width_list, height_list, padding_atrr)
    objs, channel, image_idx = yolov5_post.get_result_npy()
    print(objs, channel, image_idx)
```

5.27 sail.algo_yolov5_post_cpu_opt_async

在处理器上，针对 YOLOv5 模型被加速的后处理接口，内部使用线程池的方式实现。

5.27.1 __init__

接口形式:

```
def __init__(
        self,
        shape: list[list[int]],
        network_w:int = 640,
        network_h:int = 640,
```

(下页继续)

(续上页)

```
        max_queue_size: int=20,
        use_multiclass_nms: bool=True)
```

参数说明:

- shape: list[list[int]]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- use_multiclass_nms: bool

输入参数。是否使用多类 NMS，默认为使用。

5.27.2 push_data

输入数据，支持任意 batchsize 的输入。

接口形式:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

参数说明:

- channel_idx: list[int]

输入参数。输入图像序列的通道号。

- image_idx: list[int]

输入参数。输入图像序列的编号。

- input_data: list[TensorPTRWithName],

输入参数。输入数据，包含三个输出。

- dete_threshold: list[float]

输入参数。检测阈值序列。

- nms_threshold: list[float]

输入参数。nms 阈值序列。

- ost_w: list[int]

输入参数。原始图片序列的宽。

- ost_h: list[int]

输入参数。原始图片序列的高。

- padding_attrs: list[list[int]]

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

5.27.3 get_result_npy

获取最终的检测结果

接口形式:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

5.27.4 reset_anchors

更新 anchor 尺寸.

接口形式:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

参数说明:

- anchors_new: list[list[list[int]]]

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

5.28 sail.tpu_kernel_api_yolov5_detect_out

针对 3 输出的 yolov5 模型，使用智能视觉深度学习处理器 Kernel 对后处理进行加速，目前只支持 BM1684x，且 libsophon 的版本必须不低于 0.4.6（v23.03.01）。

5.28.1 __init__

接口形式:

```
def __init__(
        self,
        device_id: int,
        shape: list[list[int]],
        network_w: int = 640,
        network_h: int = 640,
        module_file: str="/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so")
```

参数说明:

- device_id: int

输入参数。使用的设备编号。

- shape: list[list[int]]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- module_file: str

输入参数。Kernel module 文件路径，默认为”/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

5.28.2 process

处理接口。

接口形式 1:

```
def process(self,
    input_data: list[TensorPTRWithName],
    dete_threshold: float,
    nms_threshold: float,
    release_input: bool = False)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

参数说明 1:

- input_data: list[TensorPTRWithName]

输入参数。输入数据，包含三个输出。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

- release_input: bool

输入参数。释放输入的内存，默认为 False。

接口形式 2:

```
def process(self,
    input_data: dict[str, Tensor],
    dete_threshold: float,
    nms_threshold: float,
    release_input: bool = False)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

参数说明 2:

- input_data: dict[str, Tensor]

输入参数。输入数据，包含三个输出。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

- release_input: bool

输入参数。释放输入的内存，默认为 False。

返回值说明:

list[list[tuple[left, top, right, bottom, class_id, score]]]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

5.28.3 reset_anchors

更新 anchor 尺寸.

接口形式:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

参数说明:

- anchors_new: list[list[list[int]]]

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```
import sophon.sail as sail
import numpy as np

def get_ratio(bmimg):
    img_w = bmimg.width()
    img_h = bmimg.height()
    r_w = 640 / img_w
    r_h = 640 / img_h
    if r_h > r_w:
        tw = 640
        th = int(r_w * img_h)
        tx1 = tx2 = 0
        ty1 = int((640 - th) / 2)
        ty2 = 640 - th - ty1
    else:
        tw = int(r_h * img_w)
        th = 640
        tx1 = int((640 - tw) / 2)
        tx2 = 640 - tw - tx1
        ty1 = ty2 = 0

    ratio = (min(r_w, r_h), min(r_w, r_h))
    txy = (tx1, ty1)
    return (img_w, img_h), ratio, txy

if __name__ == '__main__':
    tpu_id = 0
    image_path = '../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg'
    decoder = sail.Decoder(image_path, True, tpu_id)
    bmodel_path = '../../../sophon-demo/sample/YOLOv5/models/BM1684X/yolov5s_v6.1_
↪3output_int8_1b.bmodel'
    handle = sail.Handle(tpu_id)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)

    resize_type = sail.sail_resize_type.BM_PADDING_TPU_LINEAR
    sail_engineipp = sail.EngineImagePreProcess(bmodel_path, tpu_id, False)
    ret0 = sail_engineipp.InitImagePreProcess(resize_type, True, 10, 10)

    sail_engineipp.SetPaddingAtrr(114, 114, 114, 1)
    ret1 = sail_engineipp.SetConvertAtrr(alpha_beta)

    bm_i = sail.BMImage()

    decoder.read(handle, bm_i)
    decoder.release()
```

(下页继续)

(续上页)

```
    hw, ratio, txy = get_ratio(bm_i)
    ret3 = sail_engineipp.PushImage(0, 0, bm_i)

    res = sail_engineipp.GetBatchData(True)
    output_tensor_map, ost_images, channel_list ,imageidx_list, padding_atrr = res
    tpu_kernel_3o = sail.tpu_kernel_api_yolov5_detect_out(0, [[1, 255, 80, 80], [1, 255, 40,␣
↪40], [1, 255, 20, 20]], 640, 640, "/opt/sophon/libsophon-current/lib/tpu_module/
↪libbm1684x_kernel_module.so")

    res_list = tpu_kernel_3o.process(output_tensor_map, 0.5, 0.5)

    result = []
    for i in range(len(res_list)):
        if len(res_list[i]) > 0:
            result.append(np.array(res_list[i]))
        else:
            result.append(np.empty((0,6)))

    for res in result:
        if len(res):
            coords = res[:, :4]

            coords[:, [0, 2]] -= txy[0]
            coords[:, [1, 3]] -= txy[1]
            coords[:, [0, 2]] /= ratio[0]
            coords[:, [1, 3]] /= ratio[1]

            coords[:, [0, 2]] = coords[:, [0, 2]].clip(0, hw[0] - 1)
            coords[:, [1, 3]] = coords[:, [1, 3]].clip(0, hw[1] - 1)
            res[:, :4] = coords.round()
    print(result)
```

5.29 sail.tpu_kernel_api_yolov5_out_without_decode

针对 1 输出的 yolov5 模型，使用 TPU Kernel 对后处理进行加速，目前只支持 BM1684x，且 libsophon 的版本必须不低于 0.4.6（v23.03.01）。

5.29.1 __init__

接口形式:

```
def __init__(
        self,
        device_id: int,
        shape: list[int],
        network_w: int = 640,
```

(下页继续)

(续上页)

```
        network_h: int = 640,
        module_file: str="/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so")
```

参数说明:

- device_id: int

输入参数。使用的设备编号。

- shape: list[int]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- module_file: str

输入参数。Kernel module 文件路径，默认为”/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

5.29.2 process

处理接口。

接口形式 1:

```
def process(self,
    input_data: TensorPTRWithName,
    dete_threshold: float,
    nms_threshold: float)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

参数说明 1:

- input_data: TensorPTRWithName

输入参数。输入数据，包含一个输出。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

接口形式 2:

```
def process(self,
    input_data: dict[str, Tensor],
    dete_threshold: float,
    nms_threshold: float)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

参数说明 2:

- input_data: dict[str, Tensor]

输入参数。输入数据，包含一个输出。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

返回值说明:

list[list[tuple[left, top, right, bottom, class_id, score]]]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

示例代码:

```
import sophon.sail as sail
import numpy as np

def get_ratio(bmimg):
    img_w = bmimg.width()
    img_h = bmimg.height()
    r_w = 640 / img_w
    r_h = 640 / img_h
```

(下页继续)

(续上页)

```
    if r_h > r_w:
        tw = 640
        th = int(r_w * img_h)
        tx1 = tx2 = 0
        ty1 = int((640 - th) / 2)
        ty2 = 640 - th - ty1
    else:
        tw = int(r_h * img_w)
        th = 640
        tx1 = int((640 - tw) / 2)
        tx2 = 640 - tw - tx1
        ty1 = ty2 = 0

    ratio = (min(r_w, r_h), min(r_w, r_h))
    txy = (tx1, ty1)
    return (img_w, img_h), ratio, txy

if __name__ == '__main__':
    tpu_id = 0
    image_path = '../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg'
    decoder = sail.Decoder(image_path, True, tpu_id)
    bmodel_path = '../../../sophon-demo/sample/YOLOv5/models/BM1684X/yolov5s_v6.1_
↪1output_int8_1b.bmodel'
    handle = sail.Handle(tpu_id)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)

    resize_type = sail.sail_resize_type.BM_PADDING_TPU_LINEAR
    sail_engineipp = sail.EngineImagePreProcess(bmodel_path, tpu_id, False)
    ret0 = sail_engineipp.InitImagePreProcess(resize_type, True, 10, 10)

    sail_engineipp.SetPaddingAtrr(114, 114, 114, 1)
    ret1 = sail_engineipp.SetConvertAtrr(alpha_beta)

    bm_i = sail.BMImage()

    decoder.read(handle, bm_i)
    decoder.release()
    hw, ratio, txy = get_ratio(bm_i)
    ret3 = sail_engineipp.PushImage(0, 0, bm_i)

    res = sail_engineipp.GetBatchData(True)
    output_tensor_map, ost_images, channel_list ,imageidx_list, padding_atrr = res
    tpu_kernel_1o = sail.tpu_kernel_api_yolov5_out_without_decode(0, [1, 25200, 85],␣
↪640, 640, "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so
↪")

    res_list = tpu_kernel_1o.process(output_tensor_map[0], 0.5, 0.5)

    result = []
    for i in range(len(res_list)):
        if len(res_list[i]) > 0:
```

(下页继续)

(续上页)

```
            result.append(np.array(res_list[i]))
        else:
            result.append(np.empty((0,6)))

    for res in result:
        if len(res):
            coords = res[:, :4]

            coords[:, [0, 2]] -= txy[0]
            coords[:, [1, 3]] -= txy[1]
            coords[:, [0, 2]] /= ratio[0]
            coords[:, [1, 3]] /= ratio[1]

            coords[:, [0, 2]] = coords[:, [0, 2]].clip(0, hw[0] - 1)
            coords[:, [1, 3]] = coords[:, [1, 3]].clip(0, hw[1] - 1)
            res[:, :4] = coords.round()
    print(result)
```

5.30 sort_tracker_controller

基于 SORT 算法对追踪目标进行匹配

5.30.1 __init__

接口形式:

```
def __init__(max_iou_distance:float = 0.7,max_age:int = 30, n_init:int = 3)
```

参数说明:

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

5.30.2 process

处理接口

接口形式:

```
def process(detected_objects:list[tuple[int, int, int, int, int, float]]) -> list[tuple[int, int, int,
↪int, int, float, int]]
```

参数说明:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

输入参数。检测出的物体框。

返回值说明:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

输出参数。被跟踪的物体。

示例代码:

```
import sophon.sail as sail
import cv2
import numpy as np
from python.yolov5_opencv import YOLOv5

class yolov5_arg:
    def __init__(self, bmodel, dev_id, conf_thresh, nms_thresh):
        self.bmodel = bmodel
        self.dev_id = dev_id
        self.conf_thresh = conf_thresh
        self.nms_thresh = nms_thresh
if __name__ == '__main__':
    input = "data/test_car_person_1080P.mp4"
    bmodel_detector = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel"
    bmodel_extractor = "models/BM1684X/extractor_int8_1b.bmodel"

    dev_id = 0
    conf = 0.4
    nms = 0.7

    yolov5_args = yolov5_arg(bmodel_detector, dev_id, conf, nms)
    yolov5 = YOLOv5(yolov5_args)

    cap = cv2.VideoCapture(input)
    img_batch = []

    dstc = sail.sort_tracker_controller(max_iou_distance=0.7, max_age=70, n_init=3)
    track_res_all_numpy = np.array([])

    for i in range(15):
        _, im = cap.read()
```

(下页继续)

(续上页)

```
        if im is None:
            break
        img_batch.append(im)
        results = yolov5(img_batch)
        det = results[0]

        #  The order of this API and the demo is inconsistent, and the class_id and score are reversed
        det[:, [4,5]] = det[:,[5,4]]
        img_batch.clear()

        det_tuple = [tuple(row) for row in det]

        # left, top, right, bottom, class_id, score, track_id
        track_res_numpy = dstc.process(det_tuple)
        track_res_numpy = np.array(track_res_numpy)

        if i == 0:
            track_res_all_numpy = track_res_numpy
        else:
            track_res_all_numpy = np.concatenate((track_res_all_numpy, track_res_numpy), axis=0)

    cap.release()
```

5.31 sort_tracker_controller_async

SORT 算法异步处理接口, 内部用线程实现

5.31.1 __init__

接口形式:

```
def __init__(max_iou_distance:float = 0.7, max_age:int = 30, n_init:int = 3, input_queue_size:int = 10, result_queue_size:int = 10)
```

参数说明:

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

- input_queue_size: int

输入参数。输入缓冲队列的大小

- result_queue_size: int

输入参数。结果缓冲队列的大小

5.31.2 push_data

异步处理接口, 将数据推送到内部的任务队列中, 与 get_result_npy 配合使用

接口形式 1:

```
def push_data(detected_objects:list[tuple[int, int, int, int, int, float]]) -> int
```

参数说明:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

输入参数。检测出的物体框。

- feature:sail.Tensor

输入参数。检测出的物体的特征。

返回值说明:

- int

成功返回 0，失败返回其他。

5.31.3 get_result_npy

异步处理接口, 获取追踪目标的信息, 与 push_data 配合使用

接口形式:

```
def get_result_npy() -> tracked_objects:list[tuple[int, int, int, int, int, float, int]]
```

返回值说明:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

输出参数。被跟踪的物体。

示例代码:

```
import sophon.sail as sail
import cv2
import numpy as np
from python.yolov5_opencv import YOLOv5
from feature_extractor import Extractor
class yolov5_arg:
```

(下页继续)

(续上页)

```
    def __init__(self, bmodel, dev_id, conf_thresh, nms_thresh):
        self.bmodel = bmodel
        self.dev_id = dev_id
        self.conf_thresh = conf_thresh
        self.nms_thresh = nms_thresh
if __name__ == '__main__':
    input = "data/test_car_person_1080P.mp4"
    bmodel_detector = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel"

    dev_id = 0
    conf = 0.4
    nms = 0.7

    yolov5_args = yolov5_arg(bmodel_detector, dev_id, conf, nms)
    yolov5 = YOLOv5(yolov5_args)


    cap = cv2.VideoCapture(input)
    img_batch = []

    dstc = sail.sort_tracker_controller_async(max_iou_distance=0.7, max_age=70, n_
↪init=3)

    track_res_all_numpy = np.array([])

    for i in range(15):
        _, im = cap.read()
        if im is None:
            break
        img_batch.append(im)
        results = yolov5(img_batch)
        det = results[0]


        # The order of this API and the demo is inconsistent, and the class_id and score are␣
↪reversed
        det[:, [4,5]] = det[:,[5,4]]
        img_batch.clear()

        det_tuple = [tuple(row) for row in det]

        # ------------------v numpy-----------------------
        # left, top, right, bottom, class_id, score, track_id
        ret = dstc.push_data(det_tuple)
        track_res_numpy = np.array(dstc.get_result_npy())

        if i == 0:
            track_res_all_numpy = track_res_numpy
        else:
            track_res_all_numpy = np.concatenate((track_res_all_numpy, track_res_
↪numpy), axis=0)
```

(下页继续)

(续上页)

```
cap.release()
```

5.32 deepsort_tracker_controller

针对 DeepSORT 算法，通过处理检测的结果和提取的特征，实现对目标的跟踪。

5.32.1 __init__

接口形式:

```
def __init__(max_cosine_distance:float,
        nn_budget:int,
        k_feature_dim:int,
        max_iou_distance:float = 0.7,
        max_age:int = 30,
        n_init:int = 3)
```

参数说明:

- max_cosine_distance: float

输入参数。用于相似度计算的最大余弦距离阈值。

- nn_budget: int

输入参数。用于最近邻搜索的最大数量限制。

- k_feature_dim: int

输入参数。被检测的目标的特征维度。

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

5.32.2 process

处理接口

接口形式 1:

```
def process(detected_objects:list[tuple[int, int, int, int, int, float]],
        feature:sail.Tensor)
        -> list[tuple[int, int, int, int, int, float, int]]
```

参数说明:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

输入参数。检测出的物体框。

- feature: sail.Tensor

输入参数。检测出的物体的特征。

返回值说明:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

输出参数。被跟踪的物体。

接口形式 2:

```
def process(detected_objects:list[tuple[int, int, int, int, int, float]],
        feature:list[numpy.array])
        -> list[tuple[int, int, int, int, int, float, int]]
```

参数说明:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

输入参数。检测出的物体框。

- feature: list[numpy.array]

输入参数。检测出的物体的特征。

返回值说明:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

输出参数。被跟踪的物体。

示例代码:

```
# The example code relies on sophon-demo/sample/YOLOv5/python/yolov5_opencv.py and␣
↪sophon-demo/sample/DeepSORT/python/deep_sort/deep/feature_extractor.py
import sophon.sail as sail
import cv2
import numpy as np
from python.yolov5_opencv import YOLOv5
from feature_extractor import Extractor
```

(下页继续)

(续上页)

```
class yolov5_arg:
    def __init__(self, bmodel, dev_id, conf_thresh, nms_thresh):
        self.bmodel = bmodel
        self.dev_id = dev_id
        self.conf_thresh = conf_thresh
        self.nms_thresh = nms_thresh
if __name__ == '__main__':
    input = "data/test_car_person_1080P.mp4"
    bmodel_detector = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel"
    bmodel_extractor = "models/BM1684X/extractor_int8_1b.bmodel"

    dev_id = 0
    conf = 0.4
    nms = 0.7

    yolov5_args = yolov5_arg(bmodel_detector, dev_id, conf, nms)
    yolov5 = YOLOv5(yolov5_args)
    extractor = Extractor(bmodel_extractor, dev_id)

    cap = cv2.VideoCapture(input)
    img_batch = []

    dstc = sail.deepsort_tracker_controller(max_cosine_distance=0.2, nn_budget=100, k_
↪feature_dim=extractor.output_shape[1], max_iou_distance=0.7, max_age=70, n_init=3)

    track_res_all_numpy = np.array([])

    for i in range(15):
        _, im = cap.read()
        if im is None:
            break
        img_batch.append(im)
        results = yolov5(img_batch)
        det = results[0]

        im_crops = []
        for item in det:
            x1 = int(item[0])
            y1 = int(item[1])
            x2 = int(item[2])
            y2 = int(item[3])
            im_crop = im[y1:y2, x1:x2]
            im_crops.append(im_crop)

        ext_results = extractor(im_crops)

        # The order of this API and the demo is inconsistent, and the class_id and score are␣
↪reversed
        det[:, [4,5]] = det[:,[5,4]]
        img_batch.clear()
```

(下页继续)

(续上页)

```
        det_tuple = [tuple(row) for row in det]

        # left, top, right, bottom, class_id, score, track_id
        track_res_numpy = dstc.process(det_tuple, ext_results)
        track_res_numpy = np.array(track_res_numpy)

        if i == 0:
            track_res_all_numpy = track_res_numpy
        else:
            track_res_all_numpy = np.concatenate((track_res_all_numpy, track_res_
↪numpy), axis=0)

    cap.release()
```

5.33 deepsort_tracker_controller_async

DeepSORT 算法异步处理接口, 内部用线程实现

5.33.1 __init__

接口形式:

```
def __init__(max_cosine_distance:float,
        nn_budget:int,
        k_feature_dim:int,
        max_iou_distance:float = 0.7,
        max_age:int = 30,
        n_init:int = 3,
        queue_size:int = 10)
```

参数说明:

- max_cosine_distance: float

输入参数。用于相似度计算的最大余弦距离阈值。

- nn_budget: int

输入参数。用于最近邻搜索的最大数量限制。

- k_feature_dim: int

输入参数。被检测的目标的特征维度。

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

- queue_size: int

输入参数。结果缓冲队列的大小

5.33.2 push_data

异步处理接口, 将数据推送到内部的任务队列中, 与 get_result_npy 配合使用

接口形式 1:

```
def push_data(detected_objects:list[tuple[int, int, int, int, int, float]],
              feature:sail.Tensor) -> int
```

参数说明 1:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

输入参数。检测出的物体框。

- feature:sail.Tensor

输入参数。检测出的物体的特征。

返回值说明:

- int

成功返回 0，失败返回其他。

接口形式 2:

```
def push_data(detected_objects:list[tuple[int, int, int, int, int, float]],
              feature:list[numpy.array]) -> int
```

参数说明 2:

- detected_objects: list[tuple[int, int, int, int, int, float]]

输入参数。检测出的物体框。

- feature: list[numpy.array]

输入参数。检测出的物体的特征。

返回值说明:

- int

成功返回 0，失败返回其他。

5.33.3 get_result_npy

异步处理接口, 获取追踪目标的信息, 与 push_data 配合使用

接口形式:

```
def get_result_npy() -> tracked_objects:list[tuple[int, int, int, int, int, float, int]]
```

返回值说明:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

输出参数。被跟踪的物体。

示例代码:

```
# The example code relies on sophon-demo/sample/YOLOv5/python/yolov5_opencv.py and sophon-demo/sample/DeepSORT/python/deep_sort/deep/feature_extractor.py
import sophon.sail as sail
import cv2
import numpy as np
from python.yolov5_opencv import YOLOv5
from feature_extractor import Extractor
class yolov5_arg:
    def __init__(self, bmodel, dev_id, conf_thresh, nms_thresh):
        self.bmodel = bmodel
        self.dev_id = dev_id
        self.conf_thresh = conf_thresh
        self.nms_thresh = nms_thresh
if __name__ == '__main__':
    input = "data/test_car_person_1080P.mp4"
    bmodel_detector = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel"
    bmodel_extractor = "models/BM1684X/extractor_int8_1b.bmodel"

    dev_id = 0
    conf = 0.4
    nms = 0.7

    yolov5_args = yolov5_arg(bmodel_detector, dev_id, conf, nms)
    yolov5 = YOLOv5(yolov5_args)
    extractor = Extractor(bmodel_extractor, dev_id)

    cap = cv2.VideoCapture(input)
    img_batch = []

    dstc = sail.deepsort_tracker_controller_async(max_cosine_distance=0.2, nn_budget=100, k_feature_dim=extractor.output_shape[1], max_iou_distance=0.7, max_age=70, n_init=3, queue_size=10)

    track_res_all_numpy = np.array([])

    for i in range(15):
        _, im = cap.read()
```

(下页继续)

(续上页)

```
        if im is None:
            break
        img_batch.append(im)
        results = yolov5(img_batch)
        det = results[0]

        im_crops = []
        for item in det:
            x1 = int(item[0])
            y1 = int(item[1])
            x2 = int(item[2])
            y2 = int(item[3])
            im_crop = im[y1:y2, x1:x2]
            im_crops.append(im_crop)

        ext_results = extractor(im_crops)

        # The order of this API and the demo is inconsistent, and the class_id and score are␣
↪reversed
        det[:, [4,5]] = det[:,[5,4]]
        img_batch.clear()

        det_tuple = [tuple(row) for row in det]

        # ------------------v numpy------------------------
        # left, top, right, bottom, class_id, score, track_id
        ret = dstc.push_data(det_tuple, ext_results)
        track_res_numpy = np.array(dstc.get_result_npy())

        if i == 0:
            track_res_all_numpy = track_res_numpy
        else:
            track_res_all_numpy = np.concatenate((track_res_all_numpy, track_res_
↪numpy), axis=0)

    cap.release()
```

5.34 bytetrack_tracker_controller

针对 ByteTrack 算法，通过处理检测的结果，实现对目标的跟踪。

5.34.1 __init__

接口形式:

```
def __init__(frame_rate:int = 30,
        track_buffer:int = 30)
```

参数说明:

- frame_rate: int

输入参数。用于控制被追踪物体允许消失的最大帧数，数值越大则被追踪物体允许消失的最大帧数越大。

- track_buffer: int

输入参数。用于控制被追踪物体允许消失的最大帧数，数值越大则被追踪物体允许消失的最大帧数越大。

5.34.2 process

处理接口。

接口形式 1:

```
def process(detected_objects:list[list[int, float, float, float, float, float, float, float]],
        tracked_objects:list[list[int, float, float, float, float, float, float, float, int]])
           -> int
```

参数说明 1:

- detected_objects: list[list[int, float, float, float, float, float, float, float]]

输入参数。检测出的物体框。

- tracked_objects: list[list[int, float, float, float, float, float, float, float, int]]

输出参数。被跟踪的物体。

返回值说明:

int

成功返回 0，失败返回其他。

示例代码:

```
# The example code relies on sophon-demo/sample/YOLOv5/python/yolov5_opencv.py
import sophon.sail as sail
import cv2
import numpy as np
from python.yolov5_opencv import YOLOv5
class yolov5_arg:
    def __init__(self, bmodel, dev_id, conf_thresh, nms_thresh):
```

(下页继续)

(续上页)

```
        self.bmodel = bmodel
        self.dev_id = dev_id
        self.conf_thresh = conf_thresh
        self.nms_thresh = nms_thresh
if __name__ == '__main__':
    input = "datasets/test_car_person_1080P.mp4"
    bmodel = "models/BM1684X/yolov5s_v6.1_3output_int8_1b.bmodel"
    dev_id = 0
    conf = 0.4
    nms = 0.7
    yolov5_args = yolov5_arg(bmodel, dev_id, conf, nms)
    yolov5 = YOLOv5(yolov5_args)

    cap = cv2.VideoCapture(input)
    img_batch = []
    btt = sail.bytetrack_tracker_controller()
    track_res_all = np.array([])

    for i in range(50):
        _, im = cap.read()
        if im is None:
            break
        img_batch.append(im)
        results = yolov5(img_batch)
        det = results[0]

        # The order of this API and the demo is inconsistent, and the class_id and score are␣
↪reversed
        det[:, [4,5]] = det[:,[5,4]]
        det = tuple(det)
        img_batch.clear()

        det_tuple = [tuple(row) for row in det]

        # tuple(left, top, right, bottom, class_id, score, track_id)
        track_res = btt.process(det_tuple)
        track_res = np.array(track_res)
        if i == 0:
            track_res_all = track_res
        else:
            track_res_all = np.concatenate((track_res_all, track_res), axis=0)

    cap.release()
```

5.35 sail.algo_yolox_post

针对 YOLOX 模型的后处理接口，内部使用线程池的方式实现。

5.35.1 __init__

接口形式:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20)
```

参数说明:

- shape: list[int]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

5.35.2 push_npy

输入数据，只支持 batchsize 为 1 的输入，或者输入之前将数据拆分之后再送入接口。

接口形式:

```
def push_npy(self,
       channel_idx: int,
       image_idx: int,
       data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
       dete_threshold: float,
       nms_threshold: float,
       ost_w: int,
       ost_h: int,
       padding_left: int,
       padding_top: int,
       padding_width: int,
       padding_height: int) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值。

- ost_w: int

输入参数。原始图片的宽。

- ost_h: int

输入参数。原始图片的高。

- padding_left: int

输入参数。填充图像的起始点坐标 x，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_top: int

输入参数。填充图像的起始点坐标 y，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_width: int

输入参数。填充图像的宽度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_height: int

输入参数。填充图像的高度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

返回值说明:

成功返回 0，其他值表示失败。

5.35.3 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: TensorPTRWithName,
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像序列的通道号。

- image_idx: int

输入参数。输入图像序列的编号。

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值序列。

- nms_threshold: float

输入参数。nms 阈值序列。

- ost_w: int

输入参数。原始图片序列的宽。

- ost_h: int

输入参数。原始图片序列的高。

- padding_attrs: list[list[int]]

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

5.35.4 get_result_npy

获取最终的检测结果

接口形式:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg"
    bmodel_name = "../../../sophon-demo/sample/YOLOX/models/BM1684X/yolox_int8_
↪1b.bmodel"
    decoder = sail.Decoder(image_name,True,tpu_id)
    bmimg = decoder.read(handle)
    engine_image_pre_process = sail.EngineImagePreProcess(bmodel_name, tpu_id, 0)
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_
↪TPU_LINEAR, True, 10, 10)
```

(下页继续)

(续上页)

```
    engine_image_pre_process.SetPaddingAtrr(114,114,114,1)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    engine_image_pre_process.SetConvertAtrr(alpha_beta)
    ret = engine_image_pre_process.PushImage(0,0, bmimg)
    engine_image_pre_process
    output_tensor_map, ost_images, channels ,imageidxs, padding_atrr = engine_image_
↪pre_process.GetBatchData(True)
    width_list = []
    height_list= []
    for index, channel in enumerate(channels):
        width_list.append(ost_images[index].width())
        height_list.append(ost_images[index].height())
    yolox_post = sail.algo_yolox_post([[1, 3, 20, 20, 85],[1, 3, 40, 40, 85],[1, 3, 80, 80, 85]],640,
↪640,10)
    dete_thresholds = np.ones(len(channels),dtype=np.float32)
    nms_thresholds = np.ones(len(channels),dtype=np.float32)
    dete_thresholds = 0.2*dete_thresholds
    nms_thresholds = 0.5*nms_thresholds
    ret = yolox_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds,␣
↪nms_thresholds, width_list, height_list, padding_atrr)
    objs, channel, image_idx = yolox_post.get_result_npy()
    print(objs, channel, image_idx)
```

5.36 sail.algo_yolov5_post_cpu_opt

针对 3 输出或 1 输出的 yolov5 模型，对后处理进行了加速。

5.36.1 __init__

接口形式:

```
def __init__(
        self,
        shapes: list[list[int]],
        network_w: int = 640,
        network_h: int = 640)
```

参数说明:

- shapes: list[list[int]]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入高度，默认为 640。

5.36.2 process

处理接口。

接口形式 1:

```
def process(self,
    input_data: list[TensorPTRWithName],
    ost_w: list[int],
    ost_h: list[int],
    dete_threshold: list[float],
    nms_threshold: list[float],
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[list[tuple[int, int, int, int, float]]]
```

参数说明 1:

- input_data: list[TensorPTRWithName]

输入参数。输入数据，包含三个输出或一个输出。

- ost_w: list[int]

输入参数。原始图片的宽度。

- ost_h: list[int]

输入参数。原始图片的高度。

- dete_threshold: list[float]

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

- input_keep_aspect_ratio: bool

输入参数。输入图片是否保持纵横比。

- input_use_multiclass_nms: bool

输入参数。是否用多类 nms。

接口形式 2:

```
def process(self,
    input_data: dict[str, Tensor],
    ost_w: list[int],
    ost_h: list[int],
    dete_threshold: list[float],
```

(下页继续)

(续上页)

```
    nms_threshold: list[float],
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

参数说明 2:

- input_data: dict[str, Tensor]

输入参数。输入数据，包含三个输出或一个输出。

- ost_w: list[int]

输入参数。原始图片的宽度。

- ost_h: list[int]

输入参数。原始图片的高度。

- dete_threshold: list[float]

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值序列。

- input_keep_aspect_ratio: bool

输入参数。输入图片是否保持纵横比。

- input_use_multiclass_nms: bool

输入参数。是否用多类 nms。

返回值说明:

list[list[tuple[left, top, right, bottom, class_id, score]]]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

5.36.3 reset_anchors

更新 anchor 尺寸.

接口形式:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

参数说明:

- anchors_new: list[list[list[int]]]

要更新的 anchor 尺寸列表.

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "../../../sophon-demo/sample/YOLOv5/datasets/test/3.jpg"
    bmodel_name = "../../../sophon-demo/sample/YOLOv5/models/BM1684X/yolov5s_v6.
↪1_3output_int8_1b.bmodel"
    decoder = sail.Decoder(image_name,True,tpu_id)
    bmimg = decoder.read(handle)
    engine_image_pre_process = sail.EngineImagePreProcess(bmodel_name, tpu_id, 0)
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_
↪TPU_LINEAR, True, 10, 10)
    engine_image_pre_process.SetPaddingAtrr(114,114,114,1)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    engine_image_pre_process.SetConvertAtrr(alpha_beta)
    ret = engine_image_pre_process.PushImage(0,0, bmimg)
    output_tensor_map, ost_images, channels ,imageidxs, padding_atrr = engine_image_
↪pre_process.GetBatchData(True)
    width_list = []
    height_list= []
    for index, channel in enumerate(channels):
        width_list.append(ost_images[index].width())
        height_list.append(ost_images[index].height())
    yolov5_post = sail.algo_yolov5_post_cpu_opt([[1, 3, 20, 20, 85],[1, 3, 40, 40, 85],[1, 3, 80,
↪ 80, 85]],640,640)
    dete_thresholds = np.ones(len(channels),dtype=np.float32)
    nms_thresholds = np.ones(len(channels),dtype=np.float32)
    dete_thresholds = 0.2*dete_thresholds
```

（下页继续）

(续上页)

```
    nms_thresholds = 0.5*nms_thresholds
    objs = yolov5_post.process(output_tensor_map, width_list, height_list, dete_thresholds,
↪ nms_thresholds, True, True)
    print(objs)
```

5.37 sail.tpu_kernel_api_openpose_part_nms

使用智能视觉深度学习处理器 Kernel 对 part nms 后处理进行加速，目前只支持 BM1684x，且 libsophon 的版本必须不低于 0.4.6（v23.03.01）。

5.37.1 __init__

接口形式:

```
def __init__(
        self,
        device_id: int,
        network_c: int,
        module_file: str="/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so")
```

参数说明:

- device_id: int

输入参数。使用的设备编号。

- network_c: int

输入参数。输入通道数，对应关键点通道的数量。

- module_file: str

输入参数。Kernel module 文件路径，默认为”/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

5.37.2 process

处理接口。

接口形式 1:

```
def process(self,
    input_data: TensorPTRWithName,
    shape: list[int],
    threshold: list[float],
    max_peak_num: list[int])
        -> tuple[list[list[int]], list[list[float]], list[list[int]]]
```

参数说明 1:

· input_data: TensorPTRWithName

输入参数。输入数据。

· shape: list[int]

输入参数。输入数据的宽和高。

· threshold: list[float]

输入参数。检测阈值序列。

· max_peak_num: list[int]

输入参数。最大被检测关键点的数量。

接口形式 2:

```
def process(self,
    input_data: dict[str, Tensor],
    shape: list[int],
    threshold: list[float],
    max_peak_num: list[int])
        -> tuple[list[list[int]], list[list[float]], list[list[int]]]
```

参数说明 2:

· input_data: dict[str, Tensor]

输入参数。输入数据。

· shape: list[int]

输入参数。输入数据的宽和高。

· threshold: list[float]

输入参数。检测阈值序列。

· max_peak_num: list[int]

输入参数。最大被检测关键点的数量。

返回值说明:

tuple[list[list[int]], list[list[float]], list[list[int]]]

· 第一个输出: list[list[int]]

在每个通道被检测的关键点数量。

· 第二个输出: list[list[float]]

所有被检测的关键点的置信度。

· 第三个输出: list[list[int]]

所有被检测的关键点的拉平坐标。

5.37.3 reset_network_c

更新关键点通道数。

接口形式:

```
def reset_network_c(self, network_c_new: int) -> int
```

参数说明:

- network_c_new: int

要更新的通道数。

返回值说明:

成功返回 0，其他值表示失败。

示例代码:

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

if __name__ == '__main__':
    tpu_id = 0
    image_path = '../../../sophon-demo/sample/OpenPose/datasets/test/3.jpg'
    decoder = sail.Decoder(image_path, True, tpu_id)
    bmodel_path = '../../../sophon-demo/sample/OpenPose/models/BM1684/pose_coco_
↪fp32_1b.bmodel'
    handle = sail.Handle(tpu_id)
    net = sail.Engine(bmodel_path, tpu_id, sail.IOMode.SYSIO)
    src_img = cv2.imdecode(np.fromfile(image_path, dtype=np.uint8), -1)
    graph_name = net.get_graph_names()[0]
    input_name = net.get_input_names(graph_name)[0]
    output_name = net.get_output_names(graph_name)[0]
    h, w, _ = src_img.shape
    net_h = net.get_input_shape(graph_name, input_name)[2]
    net_w = net.get_input_shape(graph_name, input_name)[3]
    out_h = net.get_output_shape(graph_name, output_name)[2]
    scale = min(net_h / h, net_w / w)

    resize_img = cv2.resize(src_img, (0, 0), fx=scale, fy=scale, interpolation=cv2.INTER_
↪CUBIC)
    pad_img = cv2.copyMakeBorder(resize_img,0,net_h - resize_img.shape[0],0,net_w -␣
↪resize_img.shape[1],cv2.BORDER_CONSTANT,value=(114,114,114))
    img = np.transpose((pad_img.astype('float32')-128)/255, (2, 0, 1))

    outputs = net.process(graph_name, {input_name: img})

    output = np.transpose(list(outputs.values())[0], (1, 2, 0))
    stride = net_h / out_h
    output = cv2.resize(output, (0, 0), fx=stride, fy=stride, interpolation=cv2.INTER_
↪CUBIC)
```

（下页继续）

(续上页)

```
    output = output[:resize_img.shape[0], :resize_img.shape[1], :]
    output = cv2.resize(output, (src_img.shape[1], src_img.shape[0]), interpolation=cv2.
↪INTER_CUBIC)
    part_nms_input = np.array([gaussian_filter(output[:, :, j], sigma=3) for j in␣
↪range(output.shape[-1])])
    point_num = int(net.get_output_shape(graph_name, output_name)[1] / 3) - 1
    part_nms_input = {"input1": sail.Tensor(handle, part_nms_input[:point_num][None])}
    part_nms_input['input1'].sync_s2d()
    tka_nms = sail.tpu_kernel_api_openpose_part_nms(tpu_id, point_num, "/opt/
↪sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so")
    num_result, socre_result, coor_result = tka_nms.process(part_nms_input, [src_img.
↪shape[1], src_img.shape[0]], [0.05], [96])
    print(num_result, socre_result, coor_result)
```

5.38 sail.algo_yolov8_post_1output_async

针对以单输出 YOLOv8 模型的后处理接口，内部使用线程池的方式实现。

5.38.1 __init__

接口形式:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

参数说明:

- shape: list[int]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

5.38.2 push_npy

输入数据，只支持 batchsize 为 1 的输入，或者输入之前将数据拆分之后再送入接口。

接口形式:

```
def push_npy(self,
        channel_idx: int,
        image_idx: int,
        data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
        dete_threshold: float,
        nms_threshold: float,
        ost_w: int,
        ost_h: int,
        padding_left: int,
        padding_top: int,
        padding_width: int,
        padding_height: int) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值。

- ost_w: int

输入参数。原始图片的宽。

- ost_h: int

输入参数。原始图片的高。

- padding_left: int

输入参数。填充图像的起始点坐标 x，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_top: int

输入参数。填充图像的起始点坐标 y，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_width: int

输入参数。填充图像的宽度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_height: int

输入参数。填充图像的高度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

返回值说明:

成功返回 0，其他值表示失败。

5.38.3 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像序列的通道号。

- image_idx: int

输入参数。输入图像序列的编号。

- input_data: list[TensorPTRWithName]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值序列。

- nms_threshold: float

输入参数。nms 阈值序列。

- ost_w: int

输入参数。原始图片序列的宽。

- ost_h: int

输入参数。原始图片序列的高。

- padding_attrs: list[list[int]]

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

5.38.4 get_result_npy

获取最终的检测结果

接口形式:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "../../../sophon-demo/sample/YOLOv8/datasets/test/3.jpg"
    bmodel_name = "../../../sophon-demo/sample/YOLOv8/models/BM1684X/yolov8s_v6.1_1output_int8_1b.bmodel"
    decoder = sail.Decoder(image_name,True,tpu_id)
    bmimg = decoder.read(handle)
    engine_image_pre_process = sail.EngineImagePreProcess(bmodel_name, tpu_id, 0)
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_TPU_LINEAR, True, 10, 10)
    engine_image_pre_process.SetPaddingAtrr(114,114,114,1)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    engine_image_pre_process.SetConvertAtrr(alpha_beta)
    ret = engine_image_pre_process.PushImage(0,0, bmimg)
    output_tensor_map, ost_images, channels ,imageidxs, paddding_attrs = engine_image_pre_process.GetBatchData(True)
    width_list = []
    height_list= []
    for index, channel in enumerate(channels):
        width_list.append(ost_images[index].width())
        height_list.append(ost_images[index].height())
    yolov8_post = sail.algo_yolov8_post_1output_async([1, 25200, 85],640,640,10)
    dete_thresholds = np.ones(len(channels),dtype=np.float32)
    nms_thresholds = np.ones(len(channels),dtype=np.float32)
    dete_thresholds = 0.2*dete_thresholds
    nms_thresholds = 0.5*nms_thresholds
    ret = yolov8_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds, nms_thresholds, width_list, height_list, paddding_attrs)
    # 以下是利用push_npy接口推送 numpy 数据的示例
    .. for index, channel in enumerate(channels):
    ..     ret = yolov8_post.push_npy(channel, index, output_tensor_map[index].get_data().asnumpy(), 0.2, 0.5,
    ..             ost_images[index].width(), ost_images[index].height(),
    ..             paddding_attrs[index][0], paddding_attrs[index][1], paddding_attrs[index][2],paddding_attrs[index][3])
    objs, channel, image_idx = yolov8_post.get_result_npy()
    print(objs, channel, image_idx)
```

5.39 sail.algo_yolov8_post_cpu_opt_1output_async

针对以单输出 YOLOv8 模型的后处理接口，内部使用线程池的方式实现。

5.39.1 __init__

接口形式:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

参数说明:

- shape: list[int]

输入参数。输入数据的 shape。

- network_w: int

输入参数。模型的输入宽度，默认为 640。

- network_h: int

输入参数。模型的输入宽度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

- input_use_multiclass_nms: bool

输入参数。使用多分类 NMS, 每个框具有多个类别。

- agnostic: bool

输入参数。使用不考虑类别的 NMS 算法。

5.39.2 push_npy

输入数据，只支持 batchsize 为 1 的输入，或者输入之前将数据拆分之后再送入接口。

接口形式:

```
def push_npy(self,
       channel_idx: int,
       image_idx: int,
```

(下页继续)

(续上页)

```
    data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
    dete_threshold: float,
    nms_threshold: float,
    ost_w: int,
    ost_h: int,
    padding_left: int,
    padding_top: int,
    padding_width: int,
    padding_height: int) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像的通道号。

- image_idx: int

输入参数。输入图像的编号。

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值。

- nms_threshold: float

输入参数。nms 阈值。

- ost_w: int

输入参数。原始图片的宽。

- ost_h: int

输入参数。原始图片的高。

- padding_left: int

输入参数。填充图像的起始点坐标 x，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_top: int

输入参数。填充图像的起始点坐标 y，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_width: int

输入参数。填充图像的宽度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

- padding_height: int

输入参数。填充图像的高度，参数可以通过通用预处理的接口中或者带有预处理的推理接口中获取，也可以自己计算。

返回值说明:

成功返回 0，其他值表示失败。

5.39.3 push_data

输入数据，支持 batchsize 不为 1 的输入。

接口形式:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

参数说明:

- channel_idx: int

输入参数。输入图像序列的通道号。

- image_idx: int

输入参数。输入图像序列的编号。

- input_data: list[TensorPTRWithName]

输入参数。输入数据。

- dete_threshold: float

输入参数。检测阈值序列。

- nms_threshold: float

输入参数。nms 阈值序列。

- ost_w: int

输入参数。原始图片序列的宽。

- ost_h: int

输入参数。原始图片序列的高。

- padding_attrs: list[list[int]]

输入参数。填充图像序列的属性列表，填充的起始点坐标 x、起始点坐标 y、尺度变换之后的宽度、尺度变换之后的高度。

返回值说明:

成功返回 0，其他值表示失败。

5.39.4 get_result_npy

获取最终的检测结果

接口形式:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

返回值说明: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

检测结果最左 x 坐标。

- top: int

检测结果最上 y 坐标。

- right: int

检测结果最右 x 坐标。

- bottom: int

检测结果最下 y 坐标。

- class_id: int

检测结果的类别编号。

- score: float

检测结果的分数。

- channel_idx: int

原始图像的通道号。

- image_idx: int

原始图像的编号。

示例代码:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "../../../sophon-demo/sample/YOLOv8/datasets/test/3.jpg"
    bmodel_name = "../../../sophon-demo/sample/YOLOv8/models/BM1684X/yolov8s_v6.
↪1_1output_int8_1b.bmodel"
```

(下页继续)

(续上页)

```
    decoder = sail.Decoder(image_name,True,tpu_id)
    bmimg = decoder.read(handle)
    engine_image_pre_process = sail.EngineImagePreProcess(bmodel_name, tpu_id, 0)
    engine_image_pre_process.InitImagePreProcess(sail.sail_resize_type.BM_PADDING_
↪TPU_LINEAR, True, 10, 10)
    engine_image_pre_process.SetPaddingAtrr(114,114,114,1)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    engine_image_pre_process.SetConvertAtrr(alpha_beta)
    ret = engine_image_pre_process.PushImage(0,0, bmimg)
    output_tensor_map, ost_images, channels ,imageidxs, paddding_attrs = engine_image_
↪pre_process.GetBatchData(True)
    width_list = []
    height_list= []
    for index, channel in enumerate(channels):
        width_list.append(ost_images[index].width())
        height_list.append(ost_images[index].height())
    yolov8_post = sail.algo_yolov8_post_cpu_opt_1output_async([1, 84, 8400],640,640,10)
    dete_thresholds = np.ones(len(channels),dtype=np.float32)
    nms_thresholds = np.ones(len(channels),dtype=np.float32)
    dete_thresholds = 0.2*dete_thresholds
    nms_thresholds = 0.5*nms_thresholds
    ret = yolov8_post.push_data(channels, imageidxs, output_tensor_map, dete_thresholds,
↪ nms_thresholds, width_list, height_list, paddding_attrs)
    # 以下是利用push_npy接口推送 numpy 数据的示例
    .. for index, channel in enumerate(channels):
    ..     ret = yolov8_post.push_npy(channel, index, output_tensor_map[index].get_data().
↪asnumpy(), 0.2, 0.5,
    ..             ost_images[index].width(), ost_images[index].height(),
    ..             paddding_attrs[index][0], paddding_attrs[index][1], paddding_attrs[index][2],␣
↪paddding_attrs[index][3])
    objs, channel, image_idx = yolov8_post.get_result_npy()
    print(objs, channel, image_idx)
```

CHAPTER 6

附录

6.1 获取在 X86 主机上进行交叉编译的 Python3

1. 安装下载工具 dfss

```
pip3 install dfss --upgrade
```

2. 根据版本使用 dfss 下载编译所需要的 Python3

- Python3.5

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.5.9.tar.gz
```

- Python3.6

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.6.5.tar.gz
```

- Python3.7

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.7.3.tar.gz
```

- Python3.8

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.8.2.tar.gz
```

- Python3.9

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.9.0.tar.gz
```

- Python3.10

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.10.0.tar.gz
```

- Python3.11

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.11.0.tar.gz
```