
sophon-sail 使用手册

发行版本 3.6.0

SOPHGO

2023 年 10 月 24 日

目录

1	声明	1
2	SAIL	3
2.1	SAIL	3
3	编译安装指南	6
3.1	源码目录结构	6
3.2	SAIL 的编译及安装	7
3.2.1	编译参数	7
3.2.2	编译可被 C++ 接口调用的动态库及头文件	8
3.2.3	编译可被 Python3 接口调用的 Wheel 文件	14
3.2.4	编译用户手册	21
3.3	使用 SAIL 的 Python 接口进行开发	22
3.3.1	PCIE MODE	22
3.3.2	SOC MODE	22
3.3.3	ARM PCIE MODE	23
3.4	使用 SAIL 的 C++ 接口进行开发	23
3.4.1	PCIE MODE	23
3.4.2	SOC MODE	25
3.4.3	ARM PCIE MODE	26
4	SAIL C++ API 参考	28
4.1	Basic function	28
4.1.1	get_available_tpu_num	28
4.1.2	set_print_flag	28
4.1.3	set_dump_io_flag	29
4.1.4	set_decoder_env	29
4.2	Data type	30
4.3	PaddingAttr	30
4.3.1	构造函数 PaddingAttr()	31
4.3.2	set_stx	32
4.3.3	set_sty	32
4.3.4	set_w	32
4.3.5	set_h	33
4.3.6	set_r	33
4.3.7	set_g	33
4.3.8	set_b	34

4.4	Handle	34
4.4.1	构造函数 Handle()	34
4.4.2	get_device_id	34
4.4.3	get_sn	35
4.5	IOMode	35
4.6	bmcv_resize_algorithm	36
4.7	Format	36
4.8	ImgDtype	38
4.9	Tensor	38
4.9.1	构造函数 Tensor()	39
4.9.2	shape	39
4.9.3	scale_from	40
4.9.4	scale_to	40
4.9.5	reshape	41
4.9.6	own_sys_data	41
4.9.7	own_dev_data	41
4.9.8	sync_s2d	41
4.9.9	sync_d2s	42
4.10	Engine	42
4.10.1	构造函数	42
4.10.2	get_handle	43
4.10.3	load	44
4.10.4	get_graph_names	44
4.10.5	set_io_mode	45
4.10.6	get_input_names	45
4.10.7	get_output_names	45
4.10.8	get_max_input_shapes	46
4.10.9	get_input_shape	46
4.10.10	get_max_output_shapes	47
4.10.11	get_output_shape	47
4.10.12	get_input_dtype	48
4.10.13	get_output_dtype	48
4.10.14	get_input_scale	49
4.10.15	get_output_scale	49
4.10.16	process	50
4.10.17	get_device_id	50
4.10.18	create_input_tensors_map	51
4.10.19	create_output_tensors_map	51
4.11	MultiEngine	52
4.11.1	MultiEngine	52
4.11.2	set_print_flag	52
4.11.3	set_print_time	53
4.11.4	get_device_ids	53
4.11.5	get_graph_names	53
4.11.6	get_input_names	53
4.11.7	get_output_names	54
4.11.8	get_input_shape	54

4.11.9	get_output_shape	55
4.11.10	process	55
4.12	bm_image	56
4.13	BMImage	57
4.13.1	构造函数 BMImage()	57
4.13.2	width	58
4.13.3	height	58
4.13.4	format	58
4.13.5	dtype	58
4.13.6	data	59
4.13.7	get_device_id	59
4.13.8	get_plane_num	59
4.14	BMImageArray	60
4.14.1	构造函数	60
4.14.2	copy_from	61
4.14.3	attach_from	61
4.14.4	get_device_id	61
4.15	Decoder	61
4.15.1	构造函数 Decoder()	62
4.15.2	is_opened	62
4.15.3	read	62
4.15.4	read_	63
4.15.5	get_frame_shape	63
4.15.6	release	64
4.15.7	reconnect	64
4.15.8	enable_dump	64
4.15.9	disable_dump	64
4.15.10	dump	65
4.16	Encoder	65
4.16.1	构造函数	65
4.16.2	is_opened	67
4.16.3	pic_encode	67
4.16.4	video_write	67
4.16.5	release	68
4.17	Bmcv	68
4.17.1	构造函数 Bmcv()	68
4.17.2	bm_image_to_tensor	69
4.17.3	tensor_to_bm_image	69
4.17.4	crop_and_resize	70
4.17.5	crop	72
4.17.6	resize	73
4.17.7	vpp_crop_and_resize	75
4.17.8	vpp_crop_and_resize_padding	76
4.17.9	vpp_crop	78
4.17.10	vpp_resize	79
4.17.11	vpp_resize_padding	81
4.17.12	warp	82

4.17.13	convert_to	83
4.17.14	yuv2bgr	84
4.17.15	rectangle	85
4.17.16	imwrite	86
4.17.17	get_handle	86
4.17.18	crop_and_resize_padding	86
4.17.19	rectangle_	88
4.17.20	imwrite_	89
4.17.21	convert_format	89
4.17.22	vpp_convert_format	90
4.17.23	putText	91
4.17.24	putText_	92
4.17.25	image_add_weighted	93
4.17.26	image_copy_to	94
4.17.27	image_copy_to_padding	95
4.17.28	nms	96
4.17.29	drawPoint	96
4.17.30	drawPoint_	97
4.17.31	warp_perspective	97
4.17.32	get_bm_data_type	99
4.17.33	get_bm_image_data_format	99
4.17.34	imdecode	99
4.17.35	fft	100
4.17.36	convert_yuv420p_to_gray	100
4.18	MultiDecoder	101
4.18.1	构造函数	101
4.18.2	set_read_timeout	102
4.18.3	add_channel	102
4.18.4	del_channel	102
4.18.5	clear_queue	103
4.18.6	read	103
4.18.7	read_	104
4.18.8	reconnect	105
4.18.9	get_frame_shape	105
4.18.10	set_local_flag	105
4.19	sail_resize_type	106
4.20	ImagePreProcess	107
4.20.1	构造函数 ImagePreProcess()	107
4.20.2	SetResizeImageAttr	107
4.20.3	SetPaddingAttr	108
4.20.4	SetConvertAttr	109
4.20.5	PushImage	109
4.20.6	GetBatchData	110
4.20.7	set_print_flag	110
4.21	TensorPTRWithName	111
4.22	EngineImagePreProcess	111
4.22.1	构造函数	111

4.22.2	InitImagePreProcess	112
4.22.3	SetPaddingAttr	112
4.22.4	SetConvertAttr	113
4.22.5	PushImage	113
4.22.6	GetBatchData	114
4.22.7	GetBatchData_CV	115
4.22.8	get_graph_name	115
4.22.9	get_input_width	116
4.22.10	get_input_height	116
4.22.11	get_output_names	116
4.22.12	get_output_shape	116
4.23	algo_yolov5_post_1output	117
4.23.1	构造函数	117
4.23.2	push_data	117
4.23.3	get_result_npy	118
4.24	algo_yolov5_post_3output	119
4.24.1	构造函数	119
4.24.2	push_data	120
4.24.3	get_result_npy	121
4.24.4	reset_anchors	121
4.25	tpu_kernel_api_yolov5_detect_out	122
4.25.1	构造函数	122
4.25.2	process	122
4.25.3	reset_anchors	124
4.26	tpu_kernel_api_yolov5_out_without_decode	124
4.26.1	构造函数	124
4.26.2	process	125
4.27	deepsort_tracker_controller	126
4.27.1	构造函数	126
4.27.2	process	127
4.28	bytetrack_tracker_controller	127
4.28.1	__init__	128
4.28.2	process	128
5	SAIL Python API 参考	129
5.1	Basic function	129
5.1.1	get_available_tpu_num	129
5.1.2	set_print_flag	129
5.1.3	set_dump_io_flag	130
5.1.4	set_decoder_env	130
5.2	sail.Data type	131
5.3	sail.PaddingAttr	131
5.3.1	__init__	131
5.3.2	set_stx	132
5.3.3	set_sty	132
5.3.4	set_w	133
5.3.5	set_h	133

5.3.6	set_r	133
5.3.7	set_g	134
5.3.8	set_b	134
5.4	sail.Handle	134
5.4.1	__init__	134
5.4.2	get_device_id	135
5.4.3	get_sn	135
5.5	sail.IOMode	135
5.6	sail.bmcv_resize_algorithm	136
5.7	sail.Format	136
5.8	sail.ImgDtype	138
5.9	sail.Tensor	138
5.9.1	__init__	139
5.9.2	shape	140
5.9.3	asnumpy	140
5.9.4	update_data	140
5.9.5	scale_from	141
5.9.6	scale_to	141
5.9.7	reshape	141
5.9.8	own_sys_data	142
5.9.9	own_dev_data	142
5.9.10	sync_s2d	142
5.9.11	sync_d2s	143
5.10	sail.Engine	143
5.10.1	__init__	143
5.10.2	get_handle	144
5.10.3	load	144
5.10.4	get_graph_names	145
5.10.5	set_io_mode	145
5.10.6	get_input_names	146
5.10.7	get_output_names	146
5.10.8	get_max_input_shapes	146
5.10.9	get_input_shape	147
5.10.10	get_max_output_shapes	147
5.10.11	get_output_shape	148
5.10.12	get_input_dtype	148
5.10.13	get_output_dtype	149
5.10.14	get_input_scale	149
5.10.15	get_output_scale	150
5.10.16	process	150
5.10.17	get_device_id	151
5.10.18	create_input_tensors_map	151
5.10.19	create_output_tensors_map	152
5.11	sail.MultiEngine	152
5.11.1	MultiEngine	152
5.11.2	set_print_flag	153
5.11.3	set_print_time	153

5.11.4	get_device_ids	153
5.11.5	get_graph_names	154
5.11.6	get_input_names	154
5.11.7	get_output_names	154
5.11.8	get_input_shape	155
5.11.9	get_output_shape	155
5.11.10	process	156
5.12	sail.bm_image	156
5.13	sail.BMImage	157
5.13.1	__init__	157
5.13.2	width	158
5.13.3	height	158
5.13.4	format	158
5.13.5	dtype	158
5.13.6	data	159
5.13.7	get_device_id	159
5.13.8	asmat	159
5.13.9	get_plane_num	160
5.14	sail.BMImageArray	160
5.14.1	__init__	160
5.14.2	__getitem__	161
5.14.3	__setitem__	161
5.14.4	copy_from	161
5.14.5	attach_from	162
5.14.6	get_device_id	162
5.15	sail.Decoder	162
5.15.1	__init__	162
5.15.2	is_opened	163
5.15.3	read	163
5.15.4	read_	164
5.15.5	get_frame_shape	164
5.15.6	release	164
5.15.7	reconnect	165
5.15.8	enable_dump	165
5.15.9	disable_dump	165
5.15.10	dump	165
5.16	sail.Encoder	166
5.16.1	__init__	166
5.16.2	is_opened	167
5.16.3	pic_encode	167
5.16.4	video_write	168
5.16.5	release	168
5.17	sail.Bmcv	168
5.17.1	__init__	168
5.17.2	bm_image_to_tensor	169
5.17.3	tensor_to_bm_image	169
5.17.4	crop_and_resize	170

5.17.5	crop	171
5.17.6	resize	172
5.17.7	vpp_crop_and_resize	173
5.17.8	vpp_crop_and_resize_padding	174
5.17.9	vpp_crop	175
5.17.10	vpp_resize	175
5.17.11	vpp_resize_padding	176
5.17.12	warp	177
5.17.13	convert_to	177
5.17.14	yuv2bgr	178
5.17.15	rectangle	179
5.17.16	imwrite	180
5.17.17	get_handle	180
5.17.18	crop_and_resize_padding	180
5.17.19	rectangle_	181
5.17.20	imwrite_	182
5.17.21	convert_format	183
5.17.22	vpp_convert_format	184
5.17.23	putText	184
5.17.24	putText_	185
5.17.25	image_add_weighted	186
5.17.26	image_copy_to	187
5.17.27	image_copy_to_padding	188
5.17.28	nms	189
5.17.29	drawPoint	189
5.17.30	drawPoint_	190
5.17.31	warp_perspective	190
5.17.32	get_bm_data_type	191
5.17.33	get_bm_image_data_format	192
5.17.34	imdecode	192
5.17.35	fft	192
5.17.36	convert_yuv420p_to_gray	193
5.18	sail.MultiDecoder	194
5.18.1	__init__	194
5.18.2	set_read_timeout	194
5.18.3	add_channel	195
5.18.4	del_channel	195
5.18.5	clear_queue	195
5.18.6	read	196
5.18.7	read_	197
5.18.8	reconnect	197
5.18.9	get_frame_shape	198
5.18.10	set_local_flag	198
5.19	sail.sail_resize_type	198
5.20	sail.ImagePreProcess	199
5.20.1	__init__	199
5.20.2	SetResizeImageAttr	200

5.20.3	SetPaddingAttr	201
5.20.4	SetConvertAttr	201
5.20.5	PushImage	202
5.20.6	GetBatchData	202
5.20.7	set_print_flag	203
5.21	sail.TensorPTRWithName	203
5.21.1	get_name	203
5.21.2	get_data	204
5.22	sail.EngineImagePreProcess	204
5.22.1	__init__	204
5.22.2	InitImagePreProcess	204
5.22.3	SetPaddingAttr	205
5.22.4	SetConvertAttr	206
5.22.5	PushImage	206
5.22.6	GetBatchData_Npy	207
5.22.7	GetBatchData_Npy2	207
5.22.8	GetBatchData	208
5.22.9	get_graph_name	209
5.22.10	get_input_width	209
5.22.11	get_input_height	209
5.22.12	get_output_names	209
5.22.13	get_output_shape	210
5.23	sail.algo_yolov5_post_1output	210
5.23.1	__init__	210
5.23.2	push_npy	211
5.23.3	push_data	212
5.23.4	get_result_npy	213
5.24	sail.algo_yolov5_post_3output	214
5.24.1	__init__	214
5.24.2	push_data	214
5.24.3	get_result_npy	215
5.24.4	reset_anchors	216
5.25	sail.tpu_kernel_api_yolov5_detect_out	216
5.25.1	__init__	216
5.25.2	process	217
5.25.3	reset_anchors	219
5.26	sail.tpu_kernel_api_yolov5_out_without_decode	219
5.26.1	__init__	219
5.26.2	process	220
5.27	deepsort_tracker_controller	221
5.27.1	__init__	221
5.27.2	process	222
5.28	bytetrack_tracker_controller	222
5.28.1	__init__	223
5.28.2	process	223



法律声明

版权所有 © 算能 2022. 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

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

2.1 SAIL

SAIL (Sophon Artificial Intelligent Library) 是 sophon-sail 中的核心模块。SAIL 对 SophonSDK 中的 BMLib、BMDecoder、BMCV、BMRuntime 进行了封装，将 SophonSDK 中原有的“加载 bmodel 并驱动 TPU 推理”、“驱动 TPU 做图像处理”、“驱动 VPU 做图像和视频解码”等功能抽象成更为简单的 C++ 接口对外提供；并且使用 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 TPU 进行推理。一个 Engine 实例可以加载一个任意的 bmodel，自动地管理输入张量与输出张量对应的内存。

- Decoder:

使用 VPU 解码视频，JPU 解码图像，均为硬件解码。

- 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 TPU 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。

V3.6.0 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 TPU 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 TPU 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。

3.1 源码目录结构

源码的目录结构如下:

```
├── sophon-sail
│   ├── 3rdparty
│   │   ├── json
│   │   ├── prebuild
│   │   ├── pybind11
│   │   └── spdlog
│   ├── cmake                # Cmake Files
│   │   ├── BM168x_ARM_PCIE
│   │   ├── BM168x_LoongArch64
│   │   └── BM168x_SOC
│   ├── docs                # Documentation codes
│   │   ├── common
│   │   ├── source_common
│   │   └── source_zh
│   ├── include             # Includes
│   ├── python              # Wheel codes
│   │   ├── arm_pcie
│   │   ├── loongarch64
│   │   ├── pcie
│   │   └── soc
│   ├── python_wheels      # Python Wheels
│   │   ├── arm_pcie
│   │   ├── loongarch
│   │   └── soc
│   └── sample              # Sample files
```

(续下页)

(接上页)

```

|
|   |-- 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

3.2.1 编译参数

- BUILD_TYPE : 编译的类型, 目前有 pcie、soc、arm_pcie、loongarch 四种模式, 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。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 “ON” 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 “ON” 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 “ON”, 包含 python 版本 SAIL。

3.2.2 编译可被 C++ 接口调用的动态库及头文件

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 gcc-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 `gcc-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
    ├── 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。

3.2.3 编译可被 Python3 接口调用的 Wheel 文件

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-3.6.0-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-3.6.0-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-3.6.0-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-3.6.0-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, 或者从 [此链接](<http://219.142.246.77:65000/sharing/8MlSKnV8x>) 下载已经编译好的 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-3.6.0-py3-none-any.whl’

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

5. 安装 python wheel

将 ‘sophon-3.6.0-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-3.6.0-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 gcc-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, 也可以从 [此链接](http://219.142.246.77:65000/sharing/8MlSKnV8x) 下载已经编译好的 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
```

- 打包生成 python wheel, 生成的 wheel 包的路径为 ‘python/soc/dist’, 文件名为 ‘sophon_arm-3.6.0-py3-none-any.whl’

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

- 安装 python wheel

将 ‘sophon_arm-3.6.0-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-3.6.0-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, 也可以从 [此链接](<http://219.142.246.77:65000/sharing/8MlSKnV8x>) 下载已经编译好的 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。

- 下载 sophon-sail 源码, 解压后进入其源码目录
- 创建编译文件夹 build, 并进入 build 文件夹

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

- 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
```

- 打包生成 python wheel, 生成的 wheel 包的路径为 ‘python/soc/dist’, 文件名为 ‘sophon_arm-3.6.0-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-3.6.0-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-3.6.0-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 gcc-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, 也可以从 [此链接](<http://219.142.246.77:65000/sharing/8MlSKnV8x>) 下载已经编译好的 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
```

- 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-3.6.0-py3-none-any.whl’

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

- 安装 python wheel

将 ‘sophon_arm_pcie-3.6.0-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-3.6.0-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, 也可以从 [此链接](<http://219.142.246.77:65000/sharing/8MISKnV8x>) 下载已经编译好的 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。

- 下载 sophon-sail 源码, 解压后进入其源码目录
- 创建编译文件夹 build, 并进入 build 文件夹

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

- 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
```

- 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-3.6.0-py3-none-any.whl’

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

- 安装 python wheel

将 ‘sophon_arm_pcie-3.6.0-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-3.6.0-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
│   ├── 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, 也可以从 [此链接](<http://219.142.246.77:65000/sharing/8MlSKnV8x>) 下载已经编译好的 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。

- 下载 sophon-sail 源码, 解压后进入其源码目录
- 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_loongarch64-3.6.0-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_loongarch64-3.6.0-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_loongarch64-3.6.0-py3-none-any.whl --force-reinstall
```

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

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

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

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)
```

(续下页)

(接上页)

```
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_BMCMV 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::Bmcmv bmcmv(handle);

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

    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 gcc-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 gcc-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

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
```

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 TPU 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 TPU 的数量。

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。

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。

4.1.4 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

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
};
```

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

4.3 PaddingAttr

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

```
class PaddingAttr {
public:
    PaddingAttr(){};
    PaddingAttr(
        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);
    PaddingAttr(const PaddingAttr& other);
    ~PaddingAttr(){};
    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);
```

(续下页)

(接上页)

```

    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 PaddingAttr()

初始化 PaddingAttr

接口形式:

```

PaddingAttr()

PaddingAttr(
    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

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 TPU 的 id 号

4.4.2 get_device_id

获取 Handle 中 TPU 的 id

接口形式:

```
int get_device_id();
```

返回值说明:

- tpu_id: int

Handle 中的 TPU 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

4.5 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 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

4.7 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

4.8 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。

4.9 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。

4.9.1 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

4.9.2 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。

4.9.3 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

4.9.4 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。

4.9.5 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。

4.9.6 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。

4.9.7 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。

4.9.8 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。

4.9.9 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。

4.10 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。

4.10.1 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 TPU 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,
    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 TPU 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

4.10.2 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。

4.10.3 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。

4.10.4 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。

4.10.5 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。

4.10.6 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。

4.10.7 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。

4.10.8 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。

4.10.9 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。

4.10.10 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。

4.10.11 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。

4.10.12 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。

4.10.13 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。

4.10.14 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。

4.10.15 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。

4.10.16 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);

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);
```

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。

4.10.17 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。

4.10.18 create_input_tensors_map

创建输入 Tensor 的映射

接口形式:

```
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。

4.10.19 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。

4.11 MultiEngine

多线程的推理引擎，实现特定计算图的多线程推理。

4.11.1 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 TPU 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。

4.11.2 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。

4.11.3 set_print_time

设置是否打印主要处理耗时。

接口形式:

```
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。

4.11.4 get_device_ids

获取 MultiEngine 中所有可用的 TPU 的 id。

接口形式:

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

返回值说明:

- device_ids: std::vector<int>

返回可见的 TPU 的 ids

4.11.5 get_graph_names

获取 MultiEngine 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>

MultiEngine 中所有计算图的 name 的列表。

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。

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。

4.11.8 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。

4.11.9 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。

4.11.10 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, F
    ↪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。

4.12 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:

```
int width();
```

返回值说明:

- width : int

返回图像的宽。

接口形式:

```
int height();
```

返回值说明:

- height : int

返回图像的高。

接口形式:

```
int format();
```

返回值说明:

- format : bm_image_format_ext

返回图像的格式。

接口形式:

```
int dtype();
```

返回值说明:

- dtype : bm_image_data_format_ext

返回图像的数据格式。

4.13 BMImage

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

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

4.13.1 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。

4.13.2 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。

4.13.3 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。

4.13.4 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。

4.13.5 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。

4.13.6 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。

4.13.7 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

4.13.8 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 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()

4.14.1 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。

4.14.2 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。

4.14.3 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);
```

4.14.4 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

4.15 Decoder

解码器，可实现图像或视频的解码。

4.15.1 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

- tpu_id: int

设置 tpu 的 id 号。

4.15.2 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。

4.15.3 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 TPU 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。

4.15.4 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 TPU 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。

4.15.5 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。

4.15.6 release

释放 Decoder 资源。

接口形式:

```
void release();
```

4.15.7 reconnect

Decoder 再次连接。

接口形式:

```
int reconnect();
```

4.15.8 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。

4.15.9 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():  
    "" enable input video dump without encode.  
    ""
```

4.15.10 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。

4.16 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。

4.16.1 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,
        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。

- 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，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 6。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。

4.16.2 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。

4.16.3 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_PACKET, DATA_TYPE_EXT_1N_BYTE 的图片。

返回值说明:

- data: std::vector<u_char>

编码后放在系统内存中的数据。

4.16.4 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

输入参数。输入图片。在 BM1684 上，要求图片 shape 与编码器指定宽高一致，内部使用 bmcv_image_storage_convert 做格式转换。在 BM1684X 上，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。

4.16.5 release

释放编码器。

接口形式:

```
void release();
```

4.17 Bmcv

Bmcv 封装了常用的图像处理接口，支持硬件加速。

4.17.1 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。

4.17.2 bm_image_to_tensor

将 BMImage/BMImageArray 转换为 Tensor。

接口形式 1:

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

Tensor bm_image_to_tensor(BMImage &img);
```

参数说明 1:

- image: BMImage

需要转换的图像数据。

- tensor: Tensor

转换后的 Tensor。

返回值说明 1:

- tensor: Tensor

返回转换后的 Tensor。

接口形式 2:

```
def bm_image_to_tensor(self,
    image: BMImageArray,
    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。

4.17.3 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, F
    ↪ 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。

4.17.4 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_t resize_alg = BMCV_INTER_NEAREST);

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

(续下页)

(接上页)

```

    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 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。

4.17.5 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>
```

(续下页)

(接上页)

```
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。

4.17.6 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_t resize_alg = BMCV_INTER_NEAREST);
```

(续下页)

(接上页)

```

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。

4.17.7 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,
    int              crop_w,
    int              crop_h,
    int              resize_w,
    int              resize_h);

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);

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);

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);
```

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。

4.17.8 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,
    PaddingAttr  &padding_in);
```

(续下页)

(接上页)

```

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,
    PaddingAttr      &padding_in);

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,
    PaddingAttr          &padding_in);

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,
    PaddingAttr          &padding_in);

```

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 : PaddingAttr

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。

4.17.9 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);
```

(续下页)

(接上页)

```
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。

4.17.10 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,
```

(续下页)

(接上页)

```

    int                resize_h);

BMImage vpp_resize(
    BMImage            &input,
    int                resize_w,
    int                resize_h);

template<std::size_t N>
int vpp_resize(
    BMImageArray<N>    &input,
    BMImageArray<N>    &output,
    int                resize_w,
    int                resize_h);

template<std::size_t N>
BMImageArray<N> vpp_resize(
    BMImageArray<N>    &input,
    int                resize_w,
    int                resize_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。

- resize_w : int

图像 resize 的目标宽度。

- resize_h : int

图像 resize 的目标高度。

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。

4.17.11 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,
    PaddingAttr      &padding_in);

BMImage vpp_resize_padding(
    BMImage          &input,
    int              resize_w,
    int              resize_h,
    PaddingAttr      &padding_in);

template<std::size_t N>
int vpp_resize_padding(
    BMImageArray<N>    &input,
    BMImageArray<N>    &output,
    int                resize_w,
    int                resize_h,
    PaddingAttr        &padding_in);

template<std::size_t N>
BMImageArray<N> vpp_resize_padding(
    BMImageArray<N>    &input,
    int                resize_w,
    int                resize_h,
    PaddingAttr        &padding_in);
```

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 : PaddingAttr

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。

4.17.12 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);

BMImage warp(
    BMImage                &input,
    const std::pair<
        std::tuple<float, float, float>,
        std::tuple<float, float, float>> &matrix);

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);

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);
```

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。

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。

4.17.13 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 $((a_0, b_0), (a_1, b_1), (a_2, b_2))$ 。

- `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。

4.17.14 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。

4.17.15 rectangle

在图像上画一个矩形框。

接口形式:

```
int rectangle(  
    BMLImage          &image,  
    int               x0,  
    int               y0,  
    int               w,  
    int               h,  
    const std::tuple<int, int, int> &color,  
    int               thickness=1);
```

参数说明:

- image : BMLImage

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。

4.17.16 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。

4.17.17 get_handle

获取 Bmcbv 中的设备句柄 Handle。

接口形式:

```
Handle get_handle();
```

返回值说明:

- handle: Handle

Bmcbv 中的设备句柄 Handle。

4.17.18 crop_and_resize_padding

对图像进行裁剪并 resize，然后 padding。

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,
```

(续下页)

(接上页)

```

    int          resize_w,
    int          resize_h,
    PaddingAttr  &padding_in);

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,
    PaddingAttr  &padding_in);

```

参数说明:

- input : BMImage

待处理的图像。

- output : BMImage

处理后的图像。

- 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 : PaddingAttr

padding 的配置信息。

- resize_alg : bmcv_resize_algorithm

resize 采用的插值算法。

返回值说明:

- process_status : int

如果保存成功返回 0，否则返回非 0 值。

- output : BMImage

返回处理后的图像。

4.17.19 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。

4.17.20 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。

4.17.21 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。

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。

4.17.22 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。

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。

4.17.23 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。

4.17.24 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。

4.17.25 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(
    BImage      &input1,
    float       alpha,
    BImage      &input2,
    float       beta,
    float       gamma,
    BImage      &output
);
```

参数说明 1:

- input0 : BImage

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 : BImage

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: BImage

输出参数。相加后的图像 $output = input1 * alpha + input2 * beta + gamma$

接口形式 2:

```
BImage image_add_weighted(
    BImage      &input1,
    float       alpha,
    BImage      &input2,
    float       beta,
    float       gamma
);
```

参数说明 2:

- input0 : BImage

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 $output = input1 * alpha + input2 * beta + gamma$

4.17.26 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。

4.17.27 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。

4.17.28 nms

利用 TPU 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。

4.17.29 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。

4.17.30 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。

4.17.31 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>,
```

(续下页)

(接上页)

```
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。

4.17.32 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。

4.17.33 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。

4.17.34 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。

4.17.35 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。

4.17.36 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。

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。

4.18 MultiDecoder

多路解码接口，支持同时解码多路视频。

4.18.1 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 tpu 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 表示不在放数据进缓存；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。

4.18.2 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。

4.18.3 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。

4.18.4 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。

4.18.5 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。

4.18.6 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再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

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。

4.18.7 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再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

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。

4.18.8 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。

4.18.9 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。

4.18.10 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

4.19 sail_resize_type

图像预处理对应的预处理方法。

接口形式:

```
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 TPU，最近邻的方法进行图像尺度变换。

- BM_RESIZE_TPU_LINEAR

使用 TPU，线性插值的方法进行图像尺度变换。

- BM_RESIZE_TPU_BICUBIC

使用 TPU，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 TPU，最近邻的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_LINEAR

使用 TPU，线性插值的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_BICUBIC

使用 TPU，双三次插值的方法进行带 padding 的图像尺度变换。

4.20 ImagePreProcess

通用预处理接口，内部使用线程池的方式实现。

4.20.1 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 tpu 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。

4.20.2 SetResizeImageAttr

设置图像尺度变换的属性。

接口形式:

```
void SetResizeImageAttr(
    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。

4.20.3 SetPaddingAttr

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 SetPaddingAttr(
    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。

4.20.4 SetConvertAttr

设置线性变换的属性。

接口形式:

```
int SetConvertAttr(
    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。

4.20.5 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。

4.20.6 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_idx, 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_idx: 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

4.20.7 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。

4.21 TensorPTRWithName

帶有名称的 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

4.22 EngineImagePreProcess

帶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。

4.22.1 构造函数

接口形式:

```
EngineImagePreProcess(const std::string& bmodel_path,
                      int tpu_id,
                      bool use_mat_output=false);
```

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 tpu 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。

4.22.2 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。

4.22.3 SetPaddingAttr

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 SetPaddingAttr(
    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 padding 的 b 通道像素值，默认为 114。

- padding_g: int

输入参数。要 padding 的 g 通道像素值，默认为 114。

- padding_r: int

输入参数。要 padding 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。

4.22.4 SetConvertAttr

设置线性变换的属性。

接口形式:

```
int SetConvertAttr(
    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。

4.22.6 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_idxes, 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_idxes: 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。

4.22.7 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_idx, 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_idx`: `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。

4.22.8 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。

4.22.9 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。

4.22.10 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。

4.22.11 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。

4.22.12 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。

4.23 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。

4.23.1 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);
```

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。模型的输入高度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

4.23.2 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。

4.23.3 `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。

4.24 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。

4.24.1 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);
```

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。模型的输入高度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

4.24.2 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。

4.24.3 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。

4.24.4 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。

4.25 tpu_kernel_api_yolov5_detect_out

针对 3 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)。

4.25.1 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,
    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”。

4.25.2 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);
```

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。

接口形式 2:

```
std::vector<std::vector<std::tuple<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。

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。

4.25.3 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。

4.26 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)。

4.26.1 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/libsonphon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

4.26.2 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。

4.27 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。

4.27.1 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。

4.27.2 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:

int

成功返回 0，失败返回其他。

4.28 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。

4.28.1 __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。

4.28.2 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。

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 TPU 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 TPU 的数量。

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。

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。

5.1.4 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

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
```

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

5.3 sail.PaddingAttr

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

5.3.1 __init__

初始化 PaddingAttr

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

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 TPU 的 id 号

5.4.2 get_device_id

获取 Handle 中 TPU 的 id

接口形式:

```
def get_device_id(self) -> int
```

返回值说明:

- tpu_id: int

Handle 中的 TPU 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

5.5 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.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

5.7 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

5.8 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。

5.9 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。

5.9.1 `__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: tuple, 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

5.9.2 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。

5.9.3 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。

5.9.4 update_data

更新 Tensor 中系统内存的数据

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 size 不能超过 Tensor 的 size，Tensor 的 shape 将保持不变。

5.9.5 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。

5.9.6 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

5.9.7 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。

5.9.8 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。

5.9.9 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。

5.9.10 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。

5.9.11 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。

5.10 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。

5.10.1 __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 TPU 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)
```

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 TPU 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

5.10.2 `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。

5.10.3 `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) -> None
```

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)->None
```

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。

5.10.4 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。

5.10.5 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。

5.10.6 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。

5.10.7 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。

5.10.8 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。

5.10.9 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。

5.10.10 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。

5.10.11 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。

5.10.12 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。

5.10.13 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。

5.10.14 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。

5.10.15 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。

5.10.16 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})->dict {str : F
↳ 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。

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: F
↳ dict {str : sail.Tensor})->None
```

(续下页)

(接上页)

```
def process(self, graph_name: str, input_tensors: dict {str : sail.Tensor}, input_shapes: dict {str : list}, output_tensors: dict {str : sail.Tensor})->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。

5.10.17 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。

5.10.18 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)->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。

5.10.19 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) -> 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。

5.11 sail.MultiEngine

多线程的推理引擎，实现特定计算图的多线程推理。

5.11.1 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: list[int]

该 MultiEngine 可见的 TPU 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。

5.11.2 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。

5.11.3 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。

5.11.4 get_device_ids

获取 MultiEngine 中所有可用的 TPU 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 TPU 的 ids

5.11.5 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。

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。

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。

5.11.8 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。

5.11.9 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。

5.11.10 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}
```

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。

5.12 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。

5.13 sail.BMImage

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

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

5.13.1 __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。

5.13.2 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。

5.13.3 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。

5.13.4 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。

5.13.5 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。

5.13.6 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。

5.13.7 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

5.13.8 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。

5.13.9 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 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()

5.14.1 __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。

5.14.2 `__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。

5.14.3 `__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。

5.14.4 `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。

5.14.5 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
```

5.14.6 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

5.15 sail.Decoder

解码器，可实现图像或视频的解码。

5.15.1 __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

- tpu_id: int

设置 tpu 的 id 号。

5.15.2 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。

5.15.3 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 TPU 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 TPU 的 Handle。

返回值说明 2:

- image: sail.BMImage

将数据读取到 image 中。

5.15.4 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 TPU 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。

5.15.5 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。

5.15.6 release

释放 Decoder 资源。

接口形式:

```
def release(self) -> None
```

5.15.7 reconnect

Decoder 再次连接。

接口形式:

```
def reconnect(self) -> None
```

5.15.8 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。

5.15.9 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():
    """ enable input video dump without encode.
    """
```

5.15.10 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)
```

- 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。

5.16 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。

5.16.1 __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。

- enc_params: str

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, 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。

5.16.2 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。

5.16.3 pic_encode

编码一张图片, 并返回编码后的 data。

接口形式:

```
def pic_encode(self, ext: str, image: BMImage)->numpy.array
```

参数说明:

- ext: str

输入参数。图片编码格式。".jpg", ".png" 等。

- image: sail.BMImage

输入参数。输入图片, 只支持 FORMAT_BGR_PACKET, 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。

5.16.4 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:

```
def video_write(self, image: sail.BMImage)->int
```

参数说明:

- image: sail.BMImage

输入参数。输入图片。在 BM1684 上，要求图片 shape 与编码器指定宽高一致，内部使用 bmcv_image_storage_convert 做格式转换。在 BM1684X 上，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。

5.16.5 release

释放编码器。

接口形式:

```
def release(self)->None
```

5.17 sail.Bmcv

Bmcv 封装了常用的图像处理接口，支持硬件加速。

5.17.1 __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。

5.17.2 bm_image_to_tensor

将 BImage/BImageArray 转换为 Tensor。

接口形式 1:

```
def bm_image_to_tensor(self, image: sail.BImage) -> sail.Tensor
```

参数说明 1:

- image: sail.BImage

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。

接口形式 2:

```
def bm_image_to_tensor(self,  
    image: sail.BImageArray,  
    tensor) -> sail.Tensor
```

参数说明 2:

- image: sail.BImageArray

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。

5.17.3 tensor_to_bm_image

将 Tensor 转换为 BImage/BImageArray。

接口形式 1:

```
def tensor_to_bm_image(self,  
    tensor: sail.Tensor,  
    bgr2rgb: bool=False) -> sail.BImage
```

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。

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。

5.17.4 crop_and_resize

对图片进行裁剪并 resize。

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.bmcfv_resize_algorithm`

图像 `resize` 的插值算法，默认为 `sail.bmcfv_resize_algorithm.BMCFV_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。

5.17.5 crop

对图像进行裁剪。

接口形式:

```
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。

5.17.6 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。

5.17.7 vpp_crop_and_resize

利用 VPP 硬件加速图片的裁剪与 resize。

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)-> 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。

5.17.8 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:

```
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.PaddingAttr) -> 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.PaddingAttr

padding 的配置信息。

返回值说明:

- output : sail.BMImage | sail.BMImageArray

返回处理后的图像或图像数组。

5.17.9 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。

5.17.10 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)->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。

返回值说明 1:

- `output : sail.BMImage | sail.BMImageArray`

返回处理后的图像或图像数组。

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)->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。

5.17.11 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: PaddingAttr)->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.PaddingAttr`

`padding` 的配置信息。

返回值说明:

- `output : sail.BMImage | sail.BMImageArray`

返回处理后的图像或图像数组。

5.17.12 warp

对图像进行仿射变换。

接口形式:

```
def warp(self,
input: sail.BMImage | sail.BMImageArray,
matrix: list)->sail.BMImage | sail.BMImageArray
```

参数说明:

- `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。

返回值说明:

- `output : sail.BMImage | sail.BMImageArray`

返回处理后的图像或图像数组。

5.17.13 convert_to

对图像进行线性变换。

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。

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))。

5.17.14 yuv2bgr

将图像的格式从 YUV 转换为 BGR。

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。

5.17.15 rectangle

在图像上画一个矩形框。

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。

5.17.16 imwrite

将图像保存在特定文件。

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。

5.17.17 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。

5.17.18 crop_and_resize_padding

对图像进行裁剪并 resize，然后 padding。

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: PaddingAttr,
```

(续下页)

(接上页)

```
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.PaddingAttr

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。

5.17.19 rectangle_

在图像上画一个矩形框。

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。

5.17.20 imwrite_

将图像保存在指定的文件。

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。

- output : 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。

5.17.21 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:

```
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。

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。

5.17.22 vpp_convert_format

利用 VPP 硬件加速图片的格式转换。

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。

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。

5.17.23 putText

在图像上添加 text。

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。

5.17.24 putText_**接口形式:**

```
def putText_(self,
    input: sail.bm_image,
    text: str,
    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。

5.17.25 image_add_weighted

将两张图像按不同的权重相加。

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 $output = input1 * alpha + input2 * beta + gamma$

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 $output = input1 * alpha + input2 * beta + gamma$

5.17.26 image_copy_to

进行图像间的数据拷贝

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: BMLImage|BMLImageArray

输入参数。待拷贝的 BMLImage 或 BMLImageArray。

- output: BMLImage|BMLImageArray

输出参数。拷贝后的 BMLImage 或 BMLImageArray

- 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。

5.17.27 image_copy_to_padding

进行 input 和 output 间的图像数据拷贝并 padding。

接口形式:

```
def image_copy_to_padding(self,
    input: BMLImage|BMLImageArray,
    output: BMLImage|BMLImageArray,
    padding_r: int,
    padding_g: int,
    padding_b: int,
    start_x: int,
    start_y: int)->None
```

参数说明:

- input: BMLImage|BMLImageArray

输入参数。待拷贝的 BMLImage 或 BMLImageArray。

- output: BMLImage|BMLImageArray

输出参数。拷贝后的 BMLImage 或 BMLImageArray

- 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。

5.17.28 nms

利用 TPU 进行 NMS

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。

5.17.29 drawPoint

在图像上画点。

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。

5.17.30 drawPoint_

在图像上画点。

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 BMLImage 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。

5.17.31 warp_perspective

对图像进行透视变换。

接口形式:

```
def warp_perspective(self,
    input: BMLImage,
    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) -> BMLImage:
```


参数说明:

- input: BImage

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。

5.17.32 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。

5.17.33 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。

5.17.34 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。

5.17.35 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。

5.17.36 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。

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, image_format: 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。

5.18 sail.MultiDecoder

多路解码接口，支持同时解码多路视频。

5.18.1 __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 tpu 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 表示不在放数据进缓存；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。

5.18.2 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。

5.18.3 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。

5.18.4 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。

5.18.5 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。

5.18.6 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再返回。

返回值说明 1:

成功返回 0，其他值时表示失败。

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。

5.18.7 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再返回。

返回值说明 1:

成功返回 0, 其他值时表示失败。

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。

5.18.8 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。

5.18.9 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。

5.18.10 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

5.19 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 TPU，最近邻的方法进行图像尺度变换。

- BM_RESIZE_TPU_LINEAR

使用 TPU，线性插值的方法进行图像尺度变换。

- BM_RESIZE_TPU_BICUBIC

使用 TPU，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 TPU，最近邻的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_LINEAR

使用 TPU，线性插值的方法进行带 padding 的图像尺度变换。

- BM_PADDING_TPU_BICUBIC

使用 TPU，双三次插值的方法进行带 padding 的图像尺度变换。

5.20 sail.ImagePreProcess

通用预处理接口，内部使用线程池的方式实现。

5.20.1 __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 tpu 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。

5.20.2 SetResizeImageAttr

设置图像尺度变换的属性。

接口形式:

```
def SetResizeImageAttr(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。

5.20.3 SetPaddingAttr

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 SetPaddingAttr(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 padding 的 b 通道像素值，默认为 114。

- padding_g: int

输入参数。要 padding 的 g 通道像素值，默认为 114。

- padding_r: int

输入参数。要 padding 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。

5.20.4 SetConvertAttr

设置线性变换的属性。

接口形式:

```
def SetConvertAttr(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。

5.20.5 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。

5.20.6 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_idxxs, 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_idx: 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

5.20.7 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。

5.21 sail.TensorPTRWithName

带有名称的 Tensor

5.21.1 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。

5.21.2 get_data

获取 Tensor

接口形式:

```
def get_data(self) -> Tensor
```

返回值说明:

返回 Tensor

5.22 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。

5.22.1 __init__

接口形式:

```
def __init__(self,
             bmodel_path: str,
             tpu_id: int,
             use_mat_output: bool = False)
```

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 tpu 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。

5.22.2 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。

5.22.3 SetPaddingAttr

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 SetPaddingAttr(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 padding 的 b 通道像素值，默认为 114。

- `padding_g`: `int`

输入参数。要 padding 的 g 通道像素值，默认为 114。

- `padding_r`: `int`

输入参数。要 padding 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。

5.22.4 SetConvertAttr

设置线性变换的属性。

接口形式:

```
def SetConvertAttr(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。

5.22.6 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_idxes, 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_idxes: 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。

5.22.7 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_idxes, 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_idx: 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。

5.22.8 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_idx, 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_idx: 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。

5.22.9 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。

5.22.10 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。

5.22.11 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。

5.22.12 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。

5.22.13 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。

5.23 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。

5.23.1 __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。模型的输入高度，默认为 640。

- max_queue_size: int

输入参数。缓存数据的最大长度。

5.23.2 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。

5.23.3 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。

5.23.4 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。

5.24 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。

5.24.1 __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)
```

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。

5.24.2 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。

5.24.3 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。

5.24.4 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。

5.25 sail.tpu_kernel_api_yolov5_detect_out

针对 3 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)。

5.25.1 __init__

接口形式:

```
def __init__(
    self,
    device_id: int,
    shape: list[list[int]],
    network_w: int = 640,
    network_h: int = 640,
```

(续下页)

(接上页)

```
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”。

5.25.2 process

处理接口。

接口形式 1:

```
def process(self,
input_data: list[TensorPTRWithName],
dete_threshold: float,
nms_threshold: float)
-> 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。

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。

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。

5.25.3 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。

5.26 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)。

5.26.1 __init__

接口形式:

```
def __init__(
    self,
    device_id: int,
    shape: 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[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/libsonphon-current/lib/tpu_module/libbm1684x_kernel_module.so”。

5.26.2 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。

5.27 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。

5.27.1 __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。

5.27.2 process

处理接口。

接口形式 1:

```
def process(detected_objects: list[list[int, float, float, float, float, float, float, float]],
            feature: sail.Tensor,
            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。

- feature: sail.Tensor

输入参数。检测出的物体的特征。

- 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。

5.28 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。

5.28.1 __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。

5.28.2 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。