Sophon Inference Documentation

Sophon

Jan 02, 2020

CONTENTS

Technical Support

Bitmain Technologies Co., Ltd.

Address: Building 25, North Olympic Science & Technology Park, Baosheng South Road, Haidian District, Beijing, 100029, China

Website: www.sophon.ai

Email: support.ai@bitmain.com

Change History

Version	Date	Description
V2.0.1	2019.11.15	First official release
V2.0.3	2020.01.01	Add practical demos

CHAPTER

ONE

GETTING STARTED

1.1 Sophon TPU Choices

We developed four kinds of products based on our original chips. For detailed information, please refer to https://sophon.ai

1.1.1 Sophon SC serials

1.1.2 Sophon SE serials

1.1.3 Sophon SA serials

1.1.4 Sophon SM serials

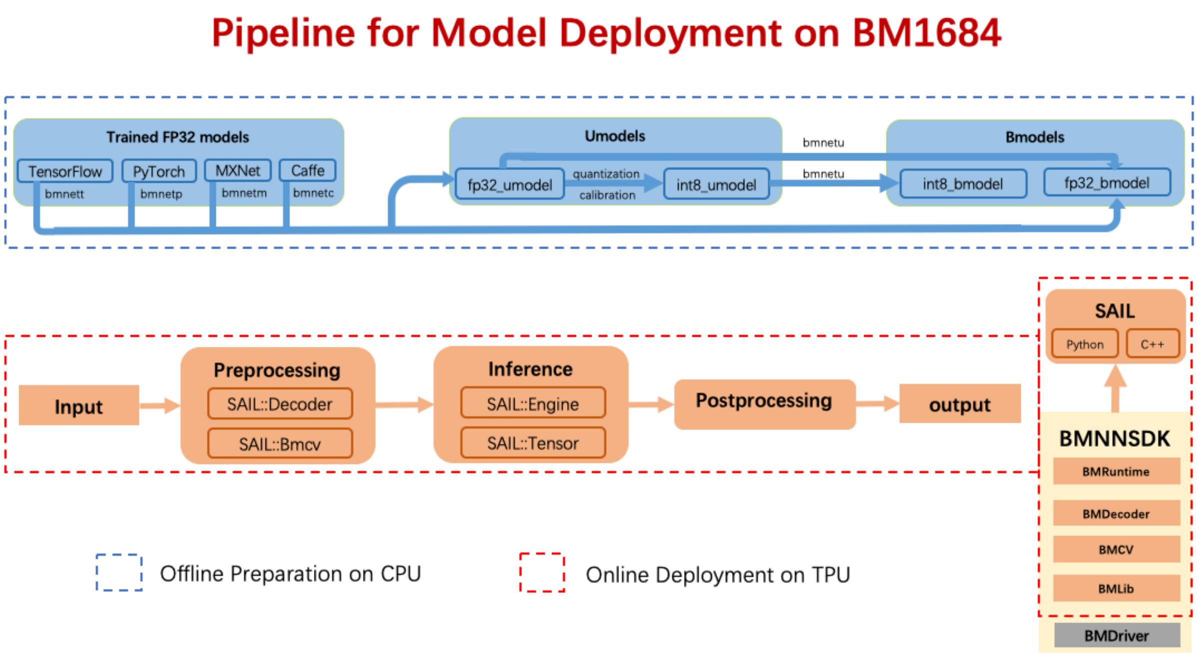
1.2 Sophon Software Stack

In response to the series of TPU products mentioned in the previous section, Bitmain independently developed a set of suitable software tools: Bitmain Neural Network Software Development Kit(BMNNSDK).

The softwares for using Sophon TPUs are all included in BMNNSDK. Sophon Inference, which supplies a bunch of high level APIs, is a upper module in BMNNSDK to help user deploying their models on Sophon TPUs rapidly.

In this section, we first show you the pipeline of deploying deep learning models on Sophon TPUs. Then, we introduce the base concepts of BMNNSDK and Sophon Inference. Last is the installation and some reminds.

1.2.1 Model deployment



Model deployment includes two steps: model offline compilation and online reasoning. Softwares shown in above picture are all included in BMNNSDK.

a).Offline Compilation

This process corresponds to the blue part in the above figure. Suppose the user has obtained a trained FP32 precision deep learning model, then the user can directly compile the model to bmodel using BMCompiler. The bmodel generated in this way can be reasoned using the FP32 computing units on the TPU. The BMCompiler is a general term here. It contains four front-end tools that support four deep learning frameworks. They are bmnetc(caffe), bmnett(tensorflow), bmnetm(mxnet), bmnetp(pytorch).

If the user wants to use the INT8 computing units on the TPU for reasoning, Quantization & Calibration tool can be used to quantify the original FP32 precision model to an INT8 precision model. Finally, user can Compile the generated int8_umodel to bmodel using the bmnetu tool in BMCompiler.

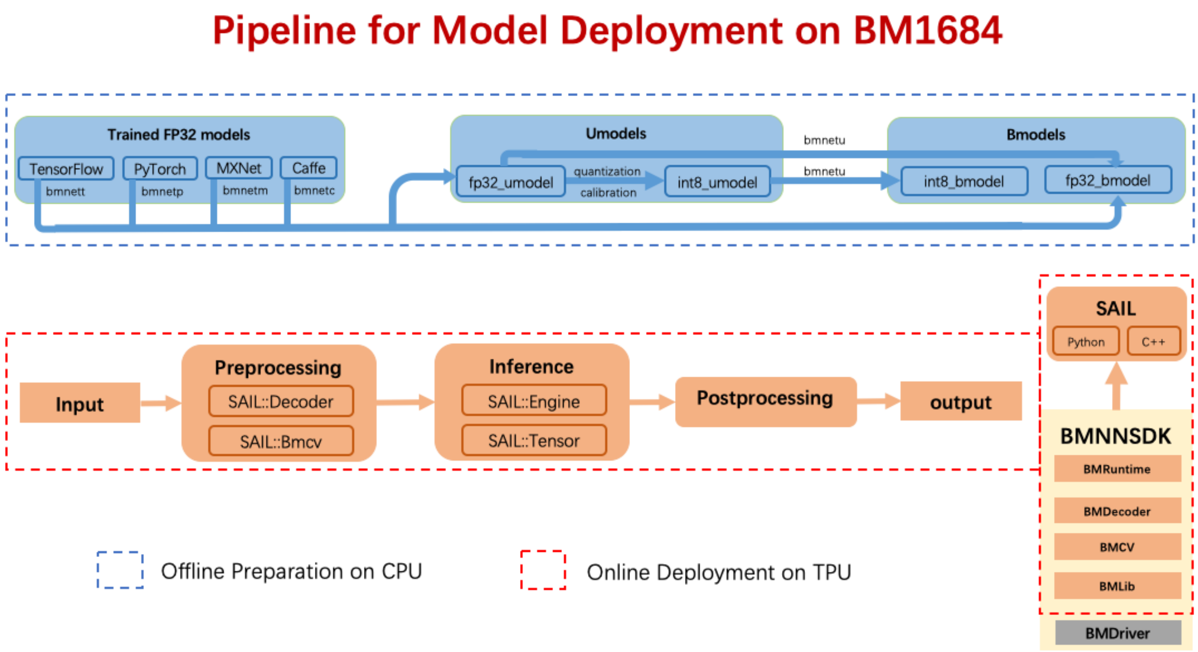
The generation of bmodel does not depend on TPU. Users only need to install the corresponding BBMCompiler and Quantization & Calibration tools as needed to complete this step. In theory, a deep learning model, as long as the bmodel can be finally generated, the bmodel can be deployed on Sophon TPUs.

b).Online Reasoning

This process corresponds to the process from input to output in the red part of the above figure. Users can do images/video decoding, tensor processing and calculations, and bmodel operations based on the SAIL module in Sophon Inference.

This process needs to be performed in the environment where the TPU and driver are installed.

1.2.2 BMNNSDK



BMNNSDK is the original deep learning development toolkit of Bitmain. It is mainly composed of modules such as Quantization & Calibration Tool, BMCompiler, BMDriver, BMLib, BMDecoder, BMCV, BMRuntime.

Quantization & Calibration Tool :It can quantize the model of FP32 precision generated by your training to INT8 precision model, which is equal to the process of converting fp32_umodel to int8_umodel in the above figure.

Online doc: https://sophon-ai-algo.github.io/calibration_tools-doc/

BMCompiler :It is a set of model compilation tools that compile your trained deep learning model into a collection of instructions that can be loaded and executed by the Sophon TPU, and save these instructions in a file with the suffix "bmodel". The tool supports compiling the FP32 model directly into bmodel. It also supports compiling the INT8 model generated by Quantization & Calibration Tool to bmodel.

Online doc: https://sophon-ai-algo.github.io/bmnnsdk-doc/

BMDriver :It is the driver for the Sophon TPU and is installed into your operating system kernel in an "insmod" manner.

BMLib :Provides basic interfaces, which can control TPU memory.

Online doc: https://sophon-ai-algo.github.io/bmlib_1684-doc/

BMDecoder :Provides interfaces which used to decode/encode image/video.

Online doc: https://sophon-ai-algo.github.io/bm_multimedia/

BMCV :It can drive TPU for image processing and tensor calculations.

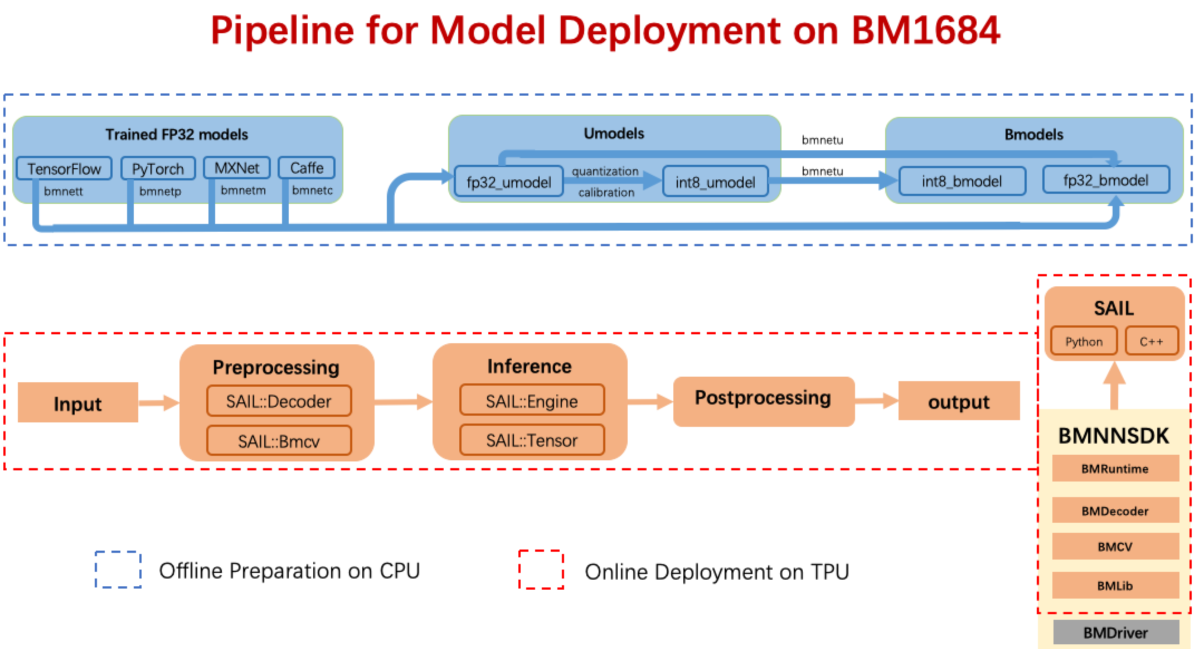
Online doc: https://sophon-ai-algo.github.io/bmcv_1684-doc/

BMRuntime :It provides interfaces to load the "bmodel" file onto the Sophon TPU and drive the TPU chip to implement reasoning.

Online doc: https://sophon-ai-algo.github.io/bmnnsdk-doc/

SAIL :It provides some high level APIs, encapsulated BMRuntime, BMCV and BMDecoder.

1.2.3 Sophon Inference



Sophon Inference currently mainly includes the SAIL(Sophon Artificial Intelligent Library) module in the above figure. We provide python/c++ interfaces and sample programs, and users can choose the appropriate calling method according to their needs.

SAIL :BMRuntime, BMCV, BMDecoder and BMLib in BMNNSDK are encapsulated. C++/python interfaces are provided. And can be used to

a).Drive the TPU to reason the compiled deep learning model (bmodel);

b).Use Sophon TPU for image and video processing.

Online English doc: https://sophon-ai-algo.github.io/sophon-inference-doc_en/

Online Chinese doc: https://sophon-ai-algo.github.io/sophon-inference-doc_zh/

1.2.4 Sophon Inference Installation

In the section of "1.1 Sophon TPU Choices" , we introduced four kinds of Sophon TPU products: SC, SE, SA, SM. SM serials are customized, which we will not mention in this article. SC serials are PCIE accelerators working on x86 platform. SE and SA serials are all SOC accelerators working on ARM platform. In SOC mode, operating system is running on the TPU memory, and schelued by the ARM core on TPU itself.

For the model deployment on SE and SA products, driving TPU to do inference is running on it, while the procedure of converting original models to bmodels are excecuted on x86 servers. And the runtime libs like BMDriver, BMRuntime, BMCV, BMdecoder are all pre-installed on SE and SA products, so, we only introduce the installation of BMMNSDK on x86 servers. If you want to deploy your application on SE and SA products finally, the installation of offline-tools in BMMNSDK is just the all you need to know.

1.2.4.1 Get BMNNSDK and Choose Link Libraries

BMNNSDK is released as a tarfile, which names as bmnnsdk2-bm1684_vx.x.x.tar.gz. bmnnsdk2 means that it is the second version serials. bm1684 represents the suitable chip version. x.x.x is the detailed version tag. We use ${BMNNSDK} as the root folder of BMNNSDK after uncompression.

Correct libraries should be choosen according to the kernel version of your system. So, a script named "install_lib.sh" should be executed once you uncomress the BMNNSDK.

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

1.2.4.2 Install Offline Tools

In the section of "1.2.2 BMNNSDK", we introduced all softwares in BMNNSDK. Offline tools includes Quantization & Calibration tool and BMCompilers. A script which should be executed once you open a new terminal is supplied for you to install all the offline tools.

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

Attention that, there are many dependencies for BMCompilers. If you only want one of the BMCompilers(bmnetc/t/m/p) or Quantization & Calibration tool, just install one of them is OK.

```
cd ${BMNNSDK}/scripts
# intall Quantization & Calibration tool
source envsetup_pcie.sh ufw
# install bmnett
source envsetup_pcie.sh bmnett
# install bmnetc
source envsetup_pcie.sh bmnetc
# install bmnetm
source envsetup_pcie.sh bmnetm
# install bmnetp
source envsetup_pcie.sh bmnetp
```

1.2.4.3 Install Runtime Tools

The runtime libraries to be installed are BMDriver and Sophon Inference by now. The installation of BMDriver needs root's priority. BMDriver will be compiler based on your kernel source and installed on system kernel after follow commands.

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

Install sophon inference:

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

1.3 Crash Course

In this course, we will help you deploy a tensorflow frozen model of mobilenet on Sophon SC5. Before starting the course, you should prepare a personal computer with a Sophon SC5 being plugined in its PCIE slot, and the BMNNSDK, Bitmain Neural Network Software Development Kit.

There are three steps in this course. First, we should install the driver, bmnett and the python module of sophon-inference. The three modules are all contained in BMNNSDK. Then, we are going to convert a mobilent which is trained from tensorflow to a bmodel using bmnett. Finally, we will deploy the converted bmodel on Sophon SC5 by sophon-inference.

1.3.1 install required softwares

BMNNSDK is the original deep learning development toolkit of Bitmain. You can contact us to get it. It is a tarfile named as bmnnsdk2-bm1684_vx.x.x.tar.gz. The bmnnsdk2 means that it is the second version of bmnnsdk. The bm1684 means that it is suitable for the chip of bm1684. After you uncompress this tarfile, all modules of bmnnsdk will placed in the folder of bmnnsdk2-bm1684_vx.x.x, which we are going to use ${BMNNSDK} to represent.

Installing libs should only be done one time after you uncompress this tarfile. The purpose of this installation is that we will choose the correct version of libs depends on your kernel version.

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

Installing driver also should only be done one time. After installing the driver, you can see "bmdev-ctl" and "bm-sophon0" under your "/dev/" path.

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

# check if installing successfully
ls /dev/ | grep "bm"
```

Installing bmnett and Configuring environment should be done as long as you open a new terminal. Bmnett is a python module and python3.5 is recommanded.

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

Sophon-Inference is a submodule of BMNNSDK which supplies a bunch of hight level APIs. With Sophon-Inference, we can rapidly deploy our deep learning models on Sophon TPU products. **Install Sophon-Inference:**

```
cd ${BMNNSDK}/exsamples/sail/python3/x86/ && pip3 install sophon-2.0.2-py3-none-any.
↪whl --user
```

1.3.2 convert tensorflow frozen model to bmodel using bmnett

We have already uploaded the official tensorflow frozen model of mobilenetv1 on our website, just "wget" it!

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

Then, convert tensorflow frozen model to bmodel using bmnett, as follow script:

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

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

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

After conversion, a bmodel named "compilation.bmodel" will generated under ${outdir}.

1.3.3 deploy bmodel on Sophon SC5 using sail

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

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

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

CHAPTER

TWO

PRACTICAL DEMOS

2.1 Preface

2.1.1 Demo Brief

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

As the above table shown, we prepared several demos to let you get familiar with Sophon-Inference more quickly. Both c++ and python are supported. For each demo, we implemented different cases to adapt to multiple applications. we have four kinds of demos by now:

cls_resnet(classification with resnet50)

det_ssd(detection with ssd300-vgg16)

det_yolov3(detection with yolov3)

det_mtcnn(detection with mtcnn)

The meanings of properties of the cases are explained as follows:

Binary: the name of binary file(c++) or script(python) of the case.

Input: input date type, image or video.

Decoder: libs for decoding the input. "opencv" is the public release version of opencv which using CPU for decoding. "bm-opencv" and "ffmpeg" are the bitmain versions of opencv and ffmpeg which using VPU for decoding.

Preprocessor: libs for processing image or tensor. "opencv" is the public release version of opencv which using CPU for caculating. "bm-opencv" and "bmcv" are the bitmain versions for processing image or tensor.

Data Type: data type of the bmodel to be used, fp32 or int8.

Model: the name of deep learning model used in this case.

Mode: two modes, static mean input tensor shapes of bmodel are unchanged, while dynamic means input tensor shapes of bmodel can be changed.

Model Number: how many models are supported concurrently in this case.

TPU Number: how many TPUs are supported at the same time.

Batch Size: the batchsize of the bmodel we used.

Multi Thread:: how many threads are supported at the same time.

2.1.2 Return Values

We also defined a return value list for each case, for reference.

ret	meaning
0	normal
1	comparing failed
2	invalid tpu id

2.2 Classification with Resnet

In this Demo, we use resnet-50 to classify images. The bmodels used in this demo are already converted from official caffe resnet-50, to both fp32 and int8 data type. We implemented four cases, they are all using public released opencv for image decoding and preprocessing. The input tensor shape of each bmodel is valid, which is the common used 1*3*224*224. The differences among the four cases are the "Model Number", "TPU Number" and "Multi-Thread".

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

2.2.1 Usage

2.2.1.1 Get model and data

To run this demo, we need both fp32 and int8 bmodels of a resnet50. We also need an image to be classified. We can get them through the script "download.py".

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

2.2.1.2 Run C++ cases

For case 0:

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

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

For case 1:

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

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

For case 2:

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

For case 3:

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

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

2.2.1.3 Run python cases

For case 0:

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

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

For case 1:

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

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

For case 2:

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

For case 3:

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

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

2.2.2 C++ Codes Explanation

2.2.2.1 Case 0: simplest case

In case 0, we encapsulated a function named "inference" , as follows:

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

The bmodel_path is the path of the bmodel of resnet50 which converted from a caffemodel of official resnet50. We use this bmodel to initialize a sail::Engine instance, for futher inference. We can get parameters, like graph_name, input_name and so on, from the sail::Engine instance.

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

Actually, you can get this information by using the "bm_model.bin" tool in BMNNSDK:

```
# fp32_bmodel
bitmain@bitmain:~$ bm_model.bin --info resnet50_fp32_191115.bmodel
# bmodel version: B.2.2
# chip: BM1684
# create time: Sat Nov 23 14:37:37 2019
```

(continues on next page)

(continued from previous page)

```
#
# ==========================================
# net: [ResNet-50_fp32]  index: [0]
# ------------
# stage: [0]  static
# input: data, [1, 3, 224, 224], float32
# output: fc1000, [1, 1000], float32

# int8_bmodel
# bitmain@bitmain:~$ bm_model.bin --info resnet50_int8_191115.bmodel
# bmodel version: B.2.2
# chip: BM1684
# create time: Sat Nov 23 14:38:50 2019
#
# ==========================================
# net: [ResNet-50_int8]  index: [0]
# ------------
# stage: [0]  static
# input: data, [1, 3, 224, 224], int8
# output: fc1000, [1, 1000], int8
```

The input_path is the path of an arbitary image. We supplied ready-made bmodels and images, the script ${sophon-inference}/tools/download.py can help you get them.

The tpu_id indicates which TPU you want to use. default value of tpu_id is 0, means using first TPU on your PC or Server.

The loops determines how many times you will run the bmodel. Let' s see what happened in the loop:

```
for (int i = 0; i < loops; ++i) {
  // read image
  cv::Mat frame = cv::imread(input_path);
  // preprocess
  preprocessor.process(input, frame);
  // scale input data if input data type is int8 or uint8
  if (in_dtype != BM_FLOAT32) {
    engine.scale_input_tensor(graph_name, input_name, input);
  }
  // inference
  engine.process(graph_name);
  // scale output data if input data type is int8 or uint8
  if (out_dtype != BM_FLOAT32) {
    engine.scale_output_tensor(graph_name, output_name, output);
  }
  // postprocess
  auto result = postprocessor.process(output);
  // print result
  for (auto item : result) {
    spdlog::info("Top 5 of loop {}: [{}]", i, fmt::join(item, ", "));
    if(!postprocessor.compare(reference, item,
        (out_dtype == BM_FLOAT32) ? "fp32" : "int8")) {
      status = false;
      break;
    }
  }
  if (!status) {
    break;
  }
}
```

As the codes shown, in each loop, we read an image from a string path to get a cv::Mat instance. Then, we do some preprocessing on the image data, like resizing. After preprocessing, we will scale the values of the data depends on its data type, this procedure is required by the int8 mode, which data should be converted from fp32 to int8 by a scale factor. Due to the pointer of the input tensor data was already stored in the SAIL::Engine instance, we only need to use the "engine.process(graph_name)" to drive bmodel to do inference. And at last, postprocessing the output tensor which data pointer was also stored in the SAIL::Engine instance. Apparently, we can execute the inference pipeline(the loop) shown above, for many times, with feeding different images.

2.2.2.2 Case 1: multi-thread implementation of case 0

In case 1, we will show the multi-thread programming mode of SAIL::Engine. Simplely, one bmodel was loaded by one SAIL::Engine instance, while input/output tensors are managed outside this SAIL::Engine instance in different threads.

We loaded the bmodel into SAIL::Engine instance after constucor, not in the constructor:

```
// init Engine
sail::Engine engine(tpu_id);
// load bmodel without builtin input and output tensors
// each thread manage its input and output tensors
int ret = engine.load(bmodel_path);
```

In each thread, we seperately managed the input and output tensors. While in case 0, these tensors were managed automatically in the SAIL::Engine instance.

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

2.2.2.3 Case 2: multi-thread with multiple models

In case 2, we will load different bmodels into a SAIL::Engine instance for each inference thread. The codes in case 2 is a little different with that in case 1. Just place the "SAIL::Engine.load(bmodel) function" into each thread is OK. In this case, we used a loading thread to finish it.

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

(continues on next page)

(continued from previous page)

```
    }
  }
```

Other codes are almost the same with case 1.

2.2.2.4 Case 3: multi-thread with multiple TPUs

In Case 3, we will exploit multiple TPUs to do inference. While the SAIL:Engine instance is bound to device, we should initialize multiple SAIL::Engine instances for each TPU.

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

Other codes are almost the same with case 1.

2.2.3 Python Codes Explanation

2.2.3.1 Case 0: simplest case

In case 0, we drive a bmodel converted from resnet50 to classfy an image. Whole procedure is composed of four steps: initializing, preprocessing, inference, postprocessing, which corresponds to four function calls.

Initializing:

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

Preprocessing:

```
image = preprocess(input_path).astype(np.float32)
```

Inference:

```
output = engine.process(graph_name, {input_name:image})
```

Postprocessing:

```
result = postprocess(output[output_name])
```

2.2.3.2 Case 1: multi-thread implementation of case 0

In case 1, we will show the multi-thread programming mode of sail.Engine. Simplely, one bmodel was loaded by one sail.Engine instance, while input/output tensors are managed outside this sail.Engine instance in different threads.

We loaded the bmodel into sail.Engine instance after constucor, not in the constructor:

```
# init Engine
engine = sail.Engine(ARGS.tpu_id)
# load bmodel without builtin input and output tensors
# each thread manage its input and output tensors
engine.load(ARGS.bmodel)
```

In each thread, we seperately managed the input and output tensors. While in case 0, these tensors were managed automatically in the sail.Engine instance.

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

2.2.3.3 Case 2: multi-thread with multiple models

In case 2, multiple bmodels could be fed. The program can create multiple threads to load different bmodels. The loading function and its caller are as follows:

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

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

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


# load bmodel without builtin input and output tensors
# each thread manage its input and output tensors
for i in range(thread_num):
  threads.append(threading.Thread(target=thread_load,
                                  args=(i, engine, ARGS.bmodel[i])))
```

Other codes are almost the same as case 1.

2.2.3.4 Case 3: multi-thread with multiple TPUs

In Case 3, we will exploit multiple TPUs to do inference. While the SAIL:Engine instance is bound to device, we should initialize multiple sail.Engine instances for each TPU.

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

Other codes are almost the same as case 1.

2.3 Detection with SSD

In this Demo, we use ssd300-vgg16 to detect objects in both images and videos. The bmodels used in this demo are already converted from official ssd300-vgg16, to both fp32 and int8 data type.

The main differences among these cases are decoder and preprocessor we choosen, except case 2, which is just the 4-N mode (batch_size is the multiples of 4) of case 1.

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

2.3.1 Usage

2.3.1.1 Get model and data

To run this demo, we need both fp32 and int8 bmodels of a ssd. We also need an image and a video to be detected. We can get them through the script "download.py".

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

2.3.1.2 Run C++ cases

For case 0:

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

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

For case 1:

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

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

For case 2:

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

For case 3:

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

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

For case 4:

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

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

2.3.1.3 Run python cases

For case 0:

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

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

For case 1:

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

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

For case 2:

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

2.3.2 C++ Codes Explanation

2.3.2.1 Case 0: decoding and preprocessing with opencv

In case 0, we exploit a bmodel converted from ssd300-vgg16 to detect objects from videos or images. We encapsulated an "inference" function to complete it. The definition of the function is:

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

(continues on next page)

(continued from previous page)

```
    int                 tpu_id,
    int                 loops,
    const std::string& compare_path);
```

In this function, we first initialize a sail::Engine instance with specified device, and load a bmodel into this sail::Engine instance.

```
// init Engine
sail::Engine engine(tpu_id);
// load bmodel without builtin input and output tensors
engine.load(bmodel_path);
```

Then, we read some parameters from this engine. Based on the information of inputs and outputs, we create tensors through sail::Tensor to hold the data.

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

We also need to initialize instances from PreProcessor, PostProcessor and CvDecoder.

For the instance of CvDecoder, we use it to decode videos or images. The CvDecoder is a virtual class defined in "cvdecoder.h", which is at the same folder of this demo. The factory method CvDecoder::create will create a decoder depends on the input path.

The PreProcessor and PostProcessor are classes defined in "processor.h", which is at the same folder of this demo. Preprocessing contains some resizing or scaling to original input tensor, while postprocessing contains bbox transformation and non-max suppression.

In the for-loop, there is a pipeline of the inference of detection:

```
// read an image from a image file or a video file
cv::Mat frame;
if (!decoder->read(frame)) {
  break;
}
// preprocess
cv::Mat img1(input_shape[2], input_shape[3], is_fp32 ? CV_32FC3 : CV_8SC3);
preprocessor.process(frame, img1);
mat_to_tensor(img1, in);
// inference
engine.process(graph_name, input_tensors, input_shapes, output_tensors);
auto real_output_shape = engine.get_output_shape(graph_name, output_name);
// postprocess
float* output_data = reinterpret_cast<float*>(out.sys_data());
std::vector<DetectRect> dets;
postprocessor.process(dets, output_data, real_output_shape,
                      frame.cols, frame.rows);
```

2.3.2.2 Case 1: decoding with bm-ffmpeg and preprocessing with bmcv

In case 1, we use bm-ffmpeg and bmcv for decoding and preprocessing. But you don't need to concern about the implementation of bm-ffmpeg and bmcv. We have already encapsulated them into SAIL.

For decoding, sail::Decoder is based on bm-ffmpeg to help you decode videos and images. Just treat sail::Decoder as cv::VideoCapture, while sail::BMImage as cv::Mat, you can easily understand the code below:

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

  // do something...

}
```

And sail::Bmcv is used for preprocessing. Other codes are almost the same with case 0.

2.3.2.3 Case 2: decoding with bm-ffmpeg and preprocessing with bmcv, 4N-mode

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

2.3.2.4 Case 3: decoding and preprocessing with bm-opencv

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

2.3.2.5 Case 4: decoding with bm-opencv and preprocessing with bmcv

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

2.3.3 Python Codes Explanation

2.3.3.1 Case 0: decoding and preprocessing with opencv

In case 0, we exploit a bmodel converted from ssd300-vgg16 to detect objects from videos or images. We encapsulated an "inference" function to complete it. The definition of the function is:

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

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

In this function, we first initialize a sail::Engine instance with specified device, and load a bmodel into this sail::Engine instance.

```
# init Engine and load bmodel
engine = sail.Engine(bmodel_path, tpu_id, sail.IOMode.SYSIO)
```

We also need to initialize instances from PreProcessor, PostProcessor and Decoder.

In this case, the decoder we used is the VideoCapture of opencv, we use it to decode videos or images.

The PreProcessor and PostProcessor are classes just defined in this script. Preprocessing contains some resizing or scaling to original input tensor, while postprocessing contains bbox transformation and non-max suppression.

```
class PreProcessor:
  """ Preprocessing class.
  """

class PostProcessor:
  """ Postprocessing class.
```

In the for-loop, there is a pipeline of the inference of detection:

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

2.3.3.2 Case 1: decoding with bm-ffmpeg and preprocessing with bmcv

In case 1, we use bm-ffmpeg and bmcv for decoding and preprocessing. But you don’t need to concern about the implementation of bm-ffmpeg and bmcv. We have already encapsulated them into SAIL.

For decoding, sail::Decoder is based on bm-ffmpeg to help you decode videos and images. Just treat sail::Decoder as cv::VideoCapture, while sail::BMImage as cv::Mat, you can easily understand the code below:

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

And sail::Bmcv is used for preprocessing. Other codes are almost the same with case 0.

2.3.3.3 Case 2: decoding with bm-ffmpeg and preprocessing with bmcv, 4N-mode

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

2.3.3.4 Case 3: decoding and preprocessing with bm-opencv

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

2.3.3.5 Case 4: decoding with bm-opencv and preprocessing with bmcv

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

2.4 Detection with Yolov3

In this Demo, we use yolov3 to detect objects in multiple videos. The bmodels used in this demo are already converted from official yolov3, to both fp32 and int8 data type.

The differences between the two cases are the Decoder and Preprocessor.

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

2.4.1 Usage

2.4.1.1 Get model and data

To run this demo, we need both fp32 and int8 bmodels of a yolov3. We also need a video to be detected. We can get them through the script "download.py".

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

2.4.1.2 Run C++ cases

For case 0:

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

For case 1:

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

2.4.1.3 Run python cases

For case 0:

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

2.4.2 C++ Codes Explanation

2.4.2.1 Case 0: decoding and preprocessing with opencv

In this case, we detect objects in multiple videos with a bmodel converted by yolov3. We use public released opencv to decode videos and process images. In the function of do_inference, we first initialize instances of sail::Engine, PreProcessor, PostProcessor:

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

// ...

PreProcessor preprocessor(416, 416);

// ...

PostProcessor postprocessor(0.5);

// ...
```

Then, we use a while-loop to process each frame of the video. The core of the pipeline are decoding, preprocessing, inference, postprocessing.

```
// ...

while (cap.read(frame)) {

  // ...

  preprocessor.processv2(input, frame);

  // ...

  engine.process(graph_name);

  // ...

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

2.4.2.2 Case 1: decoding with bm-ffmpeg and preprocessing with bmcv

In case 1, we use bm-ffmpeg and bmcv for decoding and preprocessing instead of public released opencv. Thus, FFMpegFrameProvider which is defined in "frame_provider.h" and encapsulated sail::Decoder, sail::Bmcv is used to decoding input videos. And, PreProcessorBmcv is used to processing image tensor before inference.

```
// ...

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

// ...

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

// ...

while (!frame_provider.get(img0)) {

  // ...

  preprocessor.process(img0, img1);

  // ...

  engine.process(graph_name);

  // ...

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

2.4.3 Python Codes Explanation

2.4.3.1 Case 0: decoding and preprocessing with opencv

In this case, we detect objects in multiple videos with a bmodel converted by yolov3. We use public released opencv to decode videos and process images. In the function of inference, we first initialize instances of sail::Engine:

```
# ...
net = sail.Engine(bmodel_path, tpu_id, sail.IOMode.SYSIO)

# ...
```

Then, we use a while-loop to process each frame of the input video. The core of the pipeline are decoding, preprocessing, inference, postprocessing.

```
# ...

while cap.isOpened():

  # ...

  ret, img = cap.read()

  # ...
```

(continues on next page)

(continued from previous page)

```
data = preprocess(img, detected_size)

# ...

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

# ...

output = net.process(graph_name, input_data)

# ...

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

# ...
```

2.5 Detection with MTCNN

In this Demo, we use mtcnn to detect faces in images. The bmodel used in this demo are already converted from official yolov3, to fp32 data type.

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

2.5.1 Usage

2.5.1.1 Get model and data

To run this demo, we need a fp32 bmodel of mtcnn. We also need a face image to be detected. We can get them through the script "download.py".

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

2.5.1.2 Run C++ cases

For case 0:

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

2.5.1.3 Run python cases

For case 0:

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

2.5.2 C++ Codes Explanation

2.5.2.1 Case 0

In this case, we will experience the dynamic model, mtcnn, whose input shapes is variable. There are 3 graphs in the MTCNN model: PNet, RNet and ONet. Input height and width may change for Pnet while input batch_szie may change for RNet and Onet.

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

2.5.3 Python Codes Explanation

2.5.3.1 Case 0

In this case, we will experience the dynamic model, mtcnn, whose input shapes is variable. There are 3 graphs in the MTCNN model: PNet, RNet and ONet. Input height and width may change for Pnet while input batch_szie may change for RNet and Onet.

```
# init Engine to load bmodel and allocate input and output tensors
engine = sail.Engine(bmodel_path, 0, sail.SYSIO)
# init preprocessor and postprocessor
preprocessor = PreProcessor([127.5, 127.5, 127.5], 0.0078125)
postprocessor = PostProcessor([0.5, 0.3, 0.7])
# read image
image = cv2.imread(input_path).astype(np.float32)
image = cv2.transpose(image)
# run PNet, the first stage of MTCNN
boxes = run_pnet(engine, preprocessor, postprocessor, image)
if np.array(boxes).shape[0] > 0:
  # run RNet, the second stage of MTCNN
```

(continues on next page)

(continued from previous page)

```
  boxes = run_rnet(preprocessor, postprocessor, boxes, image)
  if np.array(boxes).shape[0] > 0:
    # run ONet, the third stage of MTCNN
    boxes, points = run_onet(preprocessor, postprocessor, boxes, image)
# print detected result
for i, bbox, prob in zip(range(len(boxes)), boxes, probs):
  print("Face {} Box: {}, Score: {}".format(i, bbox, prob))
```

CHAPTER

THREE

API REFERENCE

3.1 SAIL

SAIL is the core module in the Sophon Inference.

SAIL encapsulates BMRuntime, BMDecoder, BMCV, and BMLib in BMNNSDK. It abstracts the original functions in BMNNSDK such as "loading bmodel and driving TPU reasoning", "Drive TPU for image processing", "Drive VPU for image and video decoding" into simpler C++ interfaces for external use. And it re-encapsulate with pybind11, providing the most compact Python interfaces.

Currently, all classes, enumerations, and functions in the SAIL module are in the "sail" namespace. The documentation in this module provides an in-depth look at the modules and classes in SAIL that you might use.

The core classes include:

- Handle:

The wrapper class of bm_handle_t in BMLib. Contains device handles, contextual information, used to interact with the kernel driver information of TPU.

- Tensor:

BMLib wrapper class that encapsulates management of device memroy and synchronization with system memory.

- Engine:

The wrapper class of BMRuntime, which loads bmodel and drives the TPU for reasoning. An Instance of Engine can load an arbitrary bmodel. The memory corresponding to the input tensor and the output tensor is automatically managed.

- Decoder:

Decoder to decode videos by VPU and images by JPU.

- Bmcv:

It encapsulates a series of image processing functions that can drive the TPU for image processing.

3.2 SAIL C++ API

3.2.1 Basic function

1). get_available_tpu_num

```
/** @brief Get the number of available TPUs.
 *
 *  @return Number of available TPUs.
```

(continues on next page)

(continued from previous page)

```
 */
int get_available_tpu_num();
```

3.2.2 Data type

1). bm_data_type_t

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

3.2.3 Handle

1). Handle Constructor

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

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

2). data

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

3.2.4 Tensor

1). Tensor Constructor

```
/**
 * @brief Common constructor.
 * @detail
 *  case 0: only allocate system memory
 *          (handle, shape, dtype, true, false)
 *  case 1: only allocate device memory
 *          (handle, shape, dtype, false, true)
 *  case 2: allocate system memory and device memory
 *          (handle, shape, dtype, true, true)
 *
 * @param handle       Handle instance
```

(continues on next page)

(continued from previous page)

```
 * @param shape         Shape of the tensor
 * @param own_sys_data Indicator of whether own system memory.
 * @param own_dev_data Indicator of whether own device memory.
 */
explicit Tensor(
    Handle                  handle,
    const std::vector<int>& shape,
    bm_data_type_t          dtype,
    bool                    own_sys_data,
    bool                    own_dev_data);

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

2). Tensor Assign Function

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

3). shape

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

4). dtype

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

5). reshape

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

6). own_sys_data

```
/**
 * @brief Judge if the tensor owns data in system memory.
 *
 * @return True for owns data in system memory.
```

(continues on next page)

(continued from previous page)

```
 */
bool own_sys_data();
```

7). own_dev_data

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

8). sys_data

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

9). dev_data

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

10). reset_sys_data

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

11). reset_dev_data

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

12). sync_s2d

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

/**
 * @brief Copy data from system memory to device memory with specified size.
 *
```

(continues on next page)

(continued from previous page)

```
 * @param size Byte size to be copied
 */
void sync_s2d(int size);
```

13). sync_d2s

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

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

14). free

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

3.2.5 IOMode

1). IOMode

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

3.2.6 Engine

1). Engine Constructor

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

/**
 * @brief Constructor does not load bmodel.
 *
 * @param handle Handle created elsewhere.
```

(continues on next page)

(continued from previous page)

```
 */
Engine(const Handle&   handle);

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

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

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

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

/**
 * @brief Copy constructor.
```

(continues on next page)

(continued from previous page)

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

2). Engine Assign Function

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

3). get_handle

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

4). load

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

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

5). get_graph_names

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

6). set_io_mode

```
/**
 * @brief Set IOMode for a graph.
 *
```

(continues on next page)

(continued from previous page)

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

7). get_input_names

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

8). get_output_names

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

9). get_max_input_shapes

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

10). get_input_shape

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

11). get_max_output_shapes

```
/**
 * @brief Get max shapes of output tensors in a graph.
 *
 * For static models, the max shape is fixed and it should not be changed.
```

(continues on next page)

(continued from previous page)

```
 * For dynamic models, the tensor shape should be smaller than or equal to
 * the max shape.
 *
 * @param graph_name The specified graph name
 * @return Max shape of output tensors
 */
std::map<std::string, std::vector<int>> get_max_output_shapes(
    const std::string& graph_name);
```

12). get_output_shape

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

13). get_input_dtype

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

14). get_output_dtype

```
/**
 * @brief Get data type of an output tensor. Refer to bmdef.h as following.
 *   typedef enum {
 *     BM_FLOAT32 = 0,
 *     BM_FLOAT16 = 1,
 *     BM_INT8 = 2,
 *     BM_UINT8 = 3,
 *     BM_INT16 = 4,
 *     BM_UINT16 = 5,
 *     BM_INT32 = 6,
 *     BM_UINT32 = 7
 *   } bm_data_type_t;
 *
 * @param graph_name  The specified graph name
```

(continues on next page)

(continued from previous page)

```
 * @param tensor_name The specified tensor name
 * @return Data type of the input tensor
 */
bm_data_type_t get_output_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

15). get_input_scale

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

16). get_output_scale

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

17). reshape

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

18). get_input_tensor

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

19). get_output_tensor

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

20). scale_input_tensor

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

21). scale_output_tensor

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

22). scale_fp32_to_int8

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

23). scale_int8_to_fp32

```
/**
 * @brief Scale data from int8 to float32. Only used for int8 models.
 *
 * @param src   Poniter to int8 data
 * @param dst   Poniter to float32 data
 * @param scale Value of scale
 * @param size  Size of data
```

(continues on next page)

(continued from previous page)

```
 */
void scale_int8_to_fp32(int8_t* src, float* dst, float scale, int size);
```

24). process

```
/**
 * @brief Inference with builtin input and output tensors.
 *
 * @param graph_name The specified graph name
 */
void process(const std::string& graph_name);

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

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

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

3.2.7 BMImage

1). BMImage Constructor

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

(continues on next page)

(continued from previous page)

```
/**
 * @brief The BMImage Constructor.
 *
 * @param handle A Handle instance
 * @param h      Image width
 * @param w      Image height
 * @param format Image format
 * @param dtype  Data type
 */
BMImage(
    Handle&                   handle,
    int                       h,
    int                       w,
    bm_image_format_ext       format,
    bm_image_data_format_ext dtype);
```

2). data

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

3). width

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

4). height

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

5). format

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

3.2.8 Decoder

1). Decoder Constructor

```
/**
 * @brief Constructor.
 *
```

(continues on next page)

(continued from previous page)

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

2). is_opened

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

3). read

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

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

3.2.9 Bmcv

1). Bmcv Constructor

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

2). bm_image_to_tensor

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

(continues on next page)

(continued from previous page)

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

3). tensor_to_bm_image

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

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

4). crop_and_resize

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

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

(continues on next page)

(continued from previous page)

```
BMImage crop_and_resize(
    BMImage                     &input,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h,
    int                         resize_w,
    int                         resize_h);
```

5). crop

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

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

6). resize

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

(continues on next page)

(continued from previous page)

```
    int                            resize_w,
    int                            resize_h);

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

7). vpp_crop_and_resize

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

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

8). vpp_crop

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

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

9). vpp_resize

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

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

(continues on next page)

(continued from previous page)

```
 */
BMImage vpp_resize(
    BMImage                     &input,
    int                         resize_w,
    int                         resize_h);
```

10). warp

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

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

11). convert_to

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

/**
 * @brief Applies a linear transformation to an image.
 *
 * @param input        Input image
 * @param alpha_beta   (a0, b0), (a1, b1), (a2, b2) factors
 * @return Output image
 */
```

(continues on next page)

(continued from previous page)

```
BMImage convert_to(
    BMImage                     &input,
    const std::tuple<
      std::pair<float, float>,
      std::pair<float, float>,
      std::pair<float, float>>   &alpha_beta);
```

12). yuv2bgr

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

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

13). vpp_convert

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

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

14). convert

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

(continues on next page)

(continued from previous page)

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

15). rectangle

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

16). imwrite

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

17). get_handle

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

3.3 SAIL Python API

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

3.3.1 Basic function

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

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

3.3.2 Data type

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

3.3.3 sail.Handle

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

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

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

3.3.4 sail.IOMode

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

3.3.5 sail.Tensor

1). Tensor

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

    Parameters
```

(continues on next page)

(continued from previous page)

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

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

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

2). shape

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

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

3). asnumpy

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

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

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

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

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

4). update_data

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

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

5). scale_from

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

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

6). scale_to

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

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

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

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

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

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

7). dtype

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

(continues on next page)

(continued from previous page)

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

8). reshape

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

    Parameters
    -------
    shape : list
        New shape of the tensor
    """
```

9). own_sys_data

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

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

10). own_dev_data

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

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

11). sync_s2d

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

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

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

12). sync_d2s

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

def sync_d2s(size):
```

(continues on next page)

(continued from previous page)

```
    """ Copy data from device memory to system memory with specified size.

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

3.3.6 sail.Engine

1). Engine

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

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

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

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

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

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

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

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

def __init__(bmodel_bytes, bmodel_size, tpu_id, mode):
```

(continues on next page)

(continued from previous page)

```
    """ Constructor using default input shapes with bmodel which
    loaded in memory

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

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

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

2). get_handle

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

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

3). load

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

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

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

    Parameters
    ----------
    bmodel_bytes : bytes
```

(continues on next page)

(continued from previous page)

```
        Bytes of  bmodel in system memory
    bmodel_size : int
        Bmodel byte size
    """
```

4). get_graph_names

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

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

5). set_io_mode

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

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

6). get_input_names

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

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

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

7). get_output_names

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

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

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

8). get_max_input_shapes

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

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

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

9). get_input_shape

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

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

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

10). get_max_output_shapes

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

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

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

11). get_output_shape

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

    Parameters
    ----------
    graph_name : str
```

(continues on next page)

(continued from previous page)

```
        The specified graph name
    tensor_name : str
        The specified output tensor name

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

12). get_input_dtype

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

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

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

13). get_output_dtype

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

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

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

14). get_input_scale

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

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

    Returns
    -------
    scale: float32
```

(continues on next page)

(continued from previous page)

```
        Scale of the input tensor
    """
```

15). get_output_scale

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

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

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

16). process

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
    input_shapes : dict {str : list}
        Shapes of all input tensors
    output_tensors : dict {str : sail.Tensor}
        Output tensors managed by user
    """
```

3.3.7 sail.BMImage

1). BMImage

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

2). width

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

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

3). height

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

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

4). format

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

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

3.3.8 sail.Decoder

1). Decoder

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

    Parameters
    ----------
    file_path : str
```

(continues on next page)

(continued from previous page)

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

2). is_opened

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

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

3). read

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

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

3.3.9 sail.Bmcv

1). Bmcv

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

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

2). bm_image_to_tensor

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

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

    Returns
```

(continues on next page)

(continued from previous page)

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

3). tensor_to_bm_image

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

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

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

4). crop_and_resize

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

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

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

5). crop

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

    Parameters
    ----------
    input : sail.BMImage
        Input image
    crop_x0 : int
        Start point x of the crop window
    crop_y0 : int
```

(continues on next page)

(continued from previous page)

```
        Start point y of the crop window
    crop_w : int
        Width of the crop window
    crop_h : int
        Height of the crop window

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

6). resize

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

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

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

7). vpp_crop_and_resize

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

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

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

8). vpp_crop

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

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

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

9). vpp_resize

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

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

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

10). warp

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

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

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

11). convert_to

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

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

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

12). yuv2bgr

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

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

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

13). vpp_convert

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

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

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

14). convert

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

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

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

(continues on next page)

(continued from previous page)

```
    """
```

15). rectangle

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

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

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

16). imwrite

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

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

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

17). get_handle

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

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