



canaan

K230 nncase 开发指南

版权所有©2023 北京嘉楠捷思信息技术有限公司

免责声明

您购买的产品、服务或特性等应受北京嘉楠捷思信息技术有限公司（“本公司”，下同）及其关联公司的商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，本公司不对本文档的任何陈述、信息、内容的正确性、可靠性、完整性、适销性、符合特定目的和不侵权提供任何明示或默示的声明或保证。除非另有约定，本文档仅作为使用指导参考。

由于产品版本升级或其他原因，本文档内容将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明



、“嘉楠”和其他嘉楠商标均为北京嘉楠捷思信息技术有限公司及其关联公司的商标。本文档可能提及的其他所有商标或注册商标，由各自的所有人拥有。

版权所有 © 2023 北京嘉楠捷思信息技术有限公司。保留一切权利。

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

前言

概述

本文档为 K230 nncase 的使用说明文档，提供给用户如何安装 nncase, 如何调用 compiler APIs 编译神经网络模型和 runtime APIs 编写 AI 推理程序.

读者对象

本文档（本指南）主要适用于以下人员：

- 技术支持工程师
- 软件开发工程师

缩略词定义

简称	说明
PTQ	Post-training quantization, 训练后量化
MSE	mean-square error, 均方误差

修订记录

文档版本号	修改说明	修改者	日期
V1.0	文档初版	张扬/霍成海	2023/4/7
V1.1	统一改成 word 格式，完善 ai2d	张扬/霍成海	2023/5/5

目 录

前言.....	3
概述.....	3
读者对象.....	3
缩略词定义.....	3
修订记录.....	3
1 概述.....	8
1.1 什么是 nncase.....	8
1.2 nncase 架构.....	8
1.3 开发环境.....	10
1.3.1 操作系统	10
1.3.2 软件环境	10
1.3.3 硬件环境	11
1.4 nncase 安装.....	11
2 编译模型 APIs(Python).....	11
2.1 支持的算子.....	12
2.1.1 tf-lite 算子.....	12
2.1.2 onnx 算子.....	14
2.1.3 caffe 算子.....	18
2.2 APIs.....	19
2.2.1 CompileOptions.....	19

2.2.2	ImportOptions.....	23
2.2.3	PTQTensorOptions	24
2.2.4	set_tensor_data.....	25
2.2.5	Compiler	25
2.2.6	import_tflite.....	26
2.2.7	import_onnx	27
2.2.8	import_caffe.....	27
2.2.9	use_ptq	28
2.2.10	compile	29
2.2.11	encode_tobytes	29
2.3	示例.....	30
2.3.1	编译 tflite 模型.....	30
2.3.2	编译 onnx 模型.....	32
2.3.3	编译 caffe 模型.....	36
3	模拟器 APIs(Python).....	38
3.1	APIs.....	38
3.1.1	MemoryRange.....	38
3.1.2	RuntimeTensor	39
3.1.3	from_numpy.....	40
3.1.4	copy_to	41
3.1.5	to_numpy.....	41
3.1.6	Simulator	42
3.1.7	load_model.....	43
3.1.8	get_input_desc.....	44
3.1.9	get_output_desc.....	44
3.1.10	get_input_tensor.....	45

3.1.11	set_input_tensor	45
3.1.12	get_output_tensor	46
3.1.13	set_output_tensor	47
3.1.14	run.....	47
3.2	示例.....	48
4	KPU 运行时 APIs(C++).....	50
4.1	简介.....	50
4.2	APIs.....	51
4.2.1	hrt::create	51
4.2.2	hrt::sync	52
4.2.3	interpreter::load_model	53
4.2.4	interpreter::inputs_size	54
4.2.5	interpreter::outputs_size	55
4.2.6	interpreter:: input_shape	55
4.2.7	interpreter:: output_shape.....	56
4.2.8	interpreter:: input_tensor.....	56
4.2.9	interpreter:: output_tensor	57
4.2.10	interpreter:: run.....	58
4.3	示例.....	58
5	AI2D 运行时 APIs(C++)	60
5.1	简介.....	60
5.2	APIs.....	60
5.2.1	ai2d_format.....	60
5.2.2	ai2d_interp_method	61
5.2.3	ai2d_interp_mode.....	61
5.2.4	ai2d_pad_mode	62

5.2.5	ai2d_datatype_t	62
5.2.6	ai2d_crop_param_t	63
5.2.7	ai2d_shift_param_t	64
5.2.8	ai2d_pad_param_t	65
5.2.9	ai2d_resize_param_t	65
5.2.10	ai2d_affine_param_t	66
5.2.11	ai2d_builder:: ai2d_builder	67
5.2.12	ai2d_builder:: build_schedule	68
5.2.13	ai2d_builder:: invoke	69
5.3	示例	69
5.4	注意事项	70
6	FAQ	71
6.1	安装 wheel 包时报错: "xxx.whl is not a supported wheel on this platform."	71
6.2	编译模型时报错"python: symbol lookup error"	71
6.3	上板运行 App 推理程序时报错"std::bad_alloc"	72
6.4	上板运行 App 推理程序时报错"data.size_bytes() == size = false (bool)"	72

1 概述

1.1 什么是 nncase

nncase 是一个为 AI 加速器设计的神经网络编译器, 目前支持的 target 有 cpu/K210/K510/K230 等.

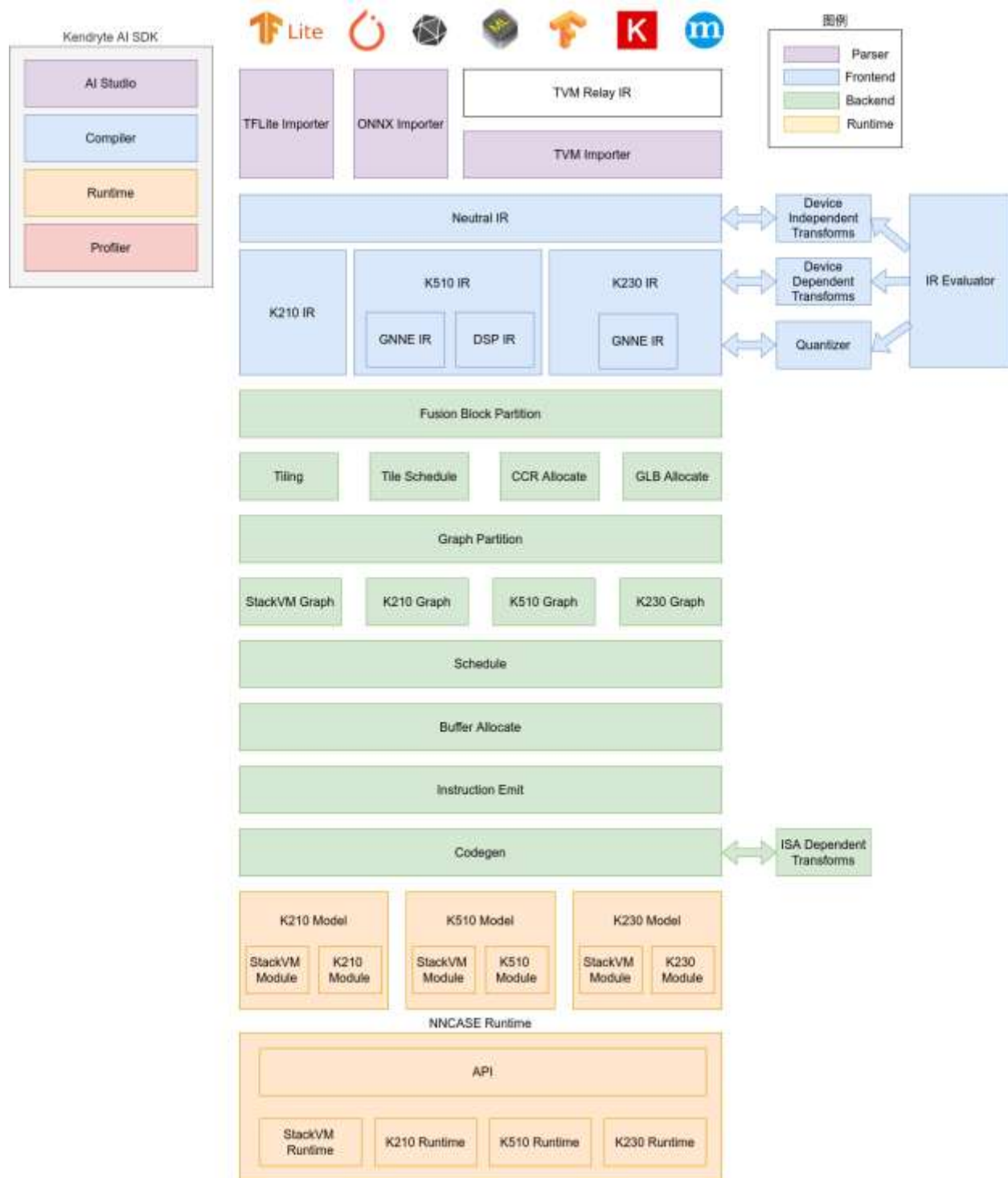
nncase 提供的功能

- 支持多输入多输出网络, 支持多分支结构
- 静态内存分配, 不需要堆内存
- 算子合并和优化
- 支持 float 和 uint8/int8 量化推理
- 支持训练后量化, 使用浮点模型和量化校准集
- 平坦模型, 支持零拷贝加载

nncase 支持的神经网络模型格式

- tflite
- onnx
- caffe

1.2 nncase 架构



nncase 软件栈包括 compiler 和 runtime 两部分。

Compiler: 用于在 PC 上编译神经网络模型，最终生成 kmodel 文件。主要包括 importer, IR, Evaluator, Quantize, Transform 优化, Tiling, Partition, Schedule, Codegen 等模块。

- Importer: 将其它神经网络框架的模型导入到 nncase 中
- IR: 中间表示, 分为 importer 导入的 Neutral IR(设备无关)和 Neutral IR 经 lowering 转换生成的 Target IR(设备相关)
- Evaluator: Evaluator 提供 IR 的解释执行能力, 常被用于 Constant Folding/PTQ Calibration 等场景
- Transform: 用于 IR 转换和图的遍历优化等
- Quantize: 训练后量化, 对要量化的 tensor 加入量化标记, 根据输入的校正集, 调用 Evaluator 进行解释执行, 收集 tensor 的数据范围, 插入量化/反量化结点, 最后优化消除不必要的量化/反量化结点等
- Tiling: 受限于 NPU 较低的存储器容量, 需要将大块计算进行拆分. 另外, 计算存在大量数据复用时选择 Tiling 参数会对时延和带宽产生影响
- Partition: 将图按 ModuleType 进行切分, 切分后的每个子图会对应 RuntimeModule, 不同类型的 RuntimeModule 对应不同的 Device(cpu/K230)
- Schedule: 根据优化后图中的数据依赖关系生成计算顺序并分配 Buffer
- Codegen: 对每个子图分别调用 ModuleType 对应的 codegen, 生成 RuntimeModule

Runtime: 集成于用户 App, 提供加载 kmodel/设置输入数据/KPU 执行/获取输出数据等功能.

1.3 开发环境

1.3.1 操作系统

支持的操作系统包括 Ubuntu 18.04/Ubuntu 20.04

1.3.2 软件环境

序号	软件	版本号
1	python	3.6/3.7/3.8/3.9/3.10
2	pip	>=20.3
3	numpy	1.19.5
4	onnx	1.9.0
5	onnx-simplifier	0.3.6
6	Onnxoptimizer	0.2.6
7	Onnxruntime	1.8.0

1.3.3 硬件环境

K230 evb

1.4 nncase 安装

nncase 工具链 compiler 部分包括 nncase 和 K230 compiler, 均需安装相应 wheel 包.

用户若没有 Ubuntu 环境, 可使用 k230_sdk 中自行编译的 k230_docker

```
$ cd /path/to/k230_sdk  
  
$ docker run -u root -it --rm -v $(pwd):/mnt -v $(pwd)/toolchain:/opt/toolchain -w /mnt k230_docker /bin/bash
```

安装 nncase(在线安装) + nncase-k230(离线安装)

```
root@c08eb1760d50:/mnt# cd src/big/ai  
  
root@c08eb1760d50:/mnt/src/big/ai# pip install nncase  
  
root@c08eb1760d50:/mnt/src/big/ai# pip install x86_64/*.whl
```

查看当前安装的 nncase 版本信息

```
root@c08eb1760d50:/mnt# python  
  
Python 3.8.10 (default, Mar 13 2023, 10:26:41)  
[GCC 9.4.0] on linux  
  
Type "help", "copyright", "credits" or "license" for more information.  
  
>>> import _nncase  
  
>>> print(_nncase.__version__)  
  
1.9.0-3473131
```

2 编译模型 APIs(Python)

nncase 提供了 Python APIs, 用于在 PC 上编译神经网络模型

2.1 支持的算子

2.1.1 tf-lite 算子

Operator	Is Supported
ABS	✓
ADD	✓
ARG_MAX	✓
ARG_MIN	✓
AVERAGE_POOL_2D	✓
BATCH_MATMUL	✓
CAST	✓
CEIL	✓
CONCATENATION	✓
CONV_2D	✓
COS	✓
CUSTOM	✓
DEPTHWISE_CONV_2D	✓
DIV	✓
EQUAL	✓
EXP	✓
EXPAND_DIMS	✓
FLOOR	✓
FLOOR_DIV	✓
FLOOR_MOD	✓
FULLY_CONNECTED	✓
GREATER	✓
GREATER_EQUAL	✓
L2_NORMALIZATION	✓
LEAKY_RELU	✓
LESS	✓
LESS_EQUAL	✓

Operator	Is Supported
LOG	✓
LOGISTIC	✓
MAX_POOL_2D	✓
MAXIMUM	✓
MEAN	✓
MINIMUM	✓
MUL	✓
NEG	✓
NOT_EQUAL	✓
PAD	✓
PADV2	✓
MIRROR_PAD	✓
PACK	✓
POW	✓
REDUCE_MAX	✓
REDUCE_MIN	✓
REDUCE_PROD	✓
RELU	✓
PRELU	✓
RELU6	✓
RESHAPE	✓
RESIZE_BILINEAR	✓
RESIZE_NEAREST_NEIGHBOR	✓
ROUND	✓
RSQRT	✓
SHAPE	✓
SIN	✓
SLICE	✓
SOFTMAX	✓
SPACE_TO_BATCH_ND	✓
SQUEEZE	✓

Operator	Is Supported
BATCH_TO_SPACE_ND	✓
STRIDED_SLICE	✓
SQRT	✓
SQUARE	✓
SUB	✓
SUM	✓
TANH	✓
TILE	✓
TRANSPOSE	✓
TRANSPOSE_CONV	✓
QUANTIZE	✓
FAKE_QUANT	✓
DEQUANTIZE	✓
GATHER	✓
GATHER_ND	✓
ONE_HOT	✓
SQUARED_DIFFERENCE	✓
LOG_SOFTMAX	✓
SPLIT	✓
HARD_SWISH	✓

2.1.2 onnx 算子

Operator	Is Supported
Abs	✓
Acos	✓
Acosh	✓
And	✓
ArgMax	✓
ArgMin	✓
Asin	✓

Operator	Is Supported
Asinh	✓
Add	✓
AveragePool	✓
BatchNormalization	✓
Cast	✓
Ceil	✓
Celu	✓
Clip	✓
Compress	✓
Concat	✓
Constant	✓
ConstantOfShape	✓
Conv	✓
ConvTranspose	✓
Cos	✓
Cosh	✓
CumSum	✓
DepthToSpace	✓
DequantizeLinear	✓
Div	✓
Dropout	✓
Elu	✓
Exp	✓
Expand	✓
Equal	✓
Erf	✓
Flatten	✓
Floor	✓
Gather	✓
GatherElements	✓
GatherND	✓

Operator	Is Supported
Gemm	✓
GlobalAveragePool	✓
GlobalMaxPool	✓
Greater	✓
GreaterOrEqual	✓
GRU	✓
Hardmax	✓
HardSigmoid	✓
HardSwish	✓
Identity	✓
InstanceNormalization	✓
LayerNormalization	✓
LpNormalization	✓
LeakyRelu	✓
Less	✓
LessOrEqual	✓
Log	✓
LogSoftmax	✓
LRN	✓
LSTM	✓
MatMul	✓
MaxPool	✓
Max	✓
Min	✓
Mul	✓
Neg	✓
Not	✓
OneHot	✓
Pad	✓
Pow	✓
PRelu	✓

Operator	Is Supported
QuantizeLinear	✓
RandomNormal	✓
RandomNormalLike	✓
RandomUniform	✓
RandomUniformLike	✓
ReduceL1	✓
ReduceL2	✓
ReduceLogSum	✓
ReduceLogSumExp	✓
ReduceMax	✓
ReduceMean	✓
ReduceMin	✓
ReduceProd	✓
ReduceSum	✓
ReduceSumSquare	✓
Relu	✓
Reshape	✓
Resize	✓
ReverseSequence	✓
RoiAlign	✓
Round	✓
Rsqrt	✓
Selu	✓
Shape	✓
Sign	✓
Sin	✓
Sinh	✓
Sigmoid	✓
Size	✓
Slice	✓
Softmax	✓

Operator	Is Supported
Softplus	✓
Softsign	✓
SpaceToDepth	✓
Split	✓
Sqrt	✓
Squeeze	✓
Sub	✓
Sum	✓
Tanh	✓
Tile	✓
TopK	✓
Transpose	✓
Trilu	✓
ThresholdedRelu	✓
Upsample	✓
Unsqueeze	✓
Where	✓

2.1.3 caffe 算子

Operator	Is Supported
Input	✓
Concat	✓
Convolution	✓
Eltwise	✓
Permute	✓
ReLU	✓
Reshape	✓
Slice	✓
Softmax	✓
Split	✓

Operator	Is Supported
ContinuationIndicator	✓
Pooling	✓
BatchNorm	✓
Scale	✓
Reverse	✓
LSTM	✓
InnerProduct	✓

2.2 APIs

目前编译模型 APIs 支持 tflite/onnx/caffe 等格式的深度学习模型。

2.2.1 CompileOptions

【描述】

CompileOptions 类, 用于配置 nncase 编译选项

【定义】

```
py::class_<compile_options>(m, "CompileOptions")
    .def(py::init())
    .def_readwrite("target", &compile_options::target)
    .def_readwrite("quant_type", &compile_options::quant_type)
    .def_readwrite("w_quant_type", &compile_options::w_quant_type)
    .def_readwrite("use_mse_quant_w", &compile_options::use_mse_quant_w)
    .def_readwrite("split_w_to_act", &compile_options::split_w_to_act)
    .def_readwrite("preprocess", &compile_options::preprocess)
    .def_readwrite("swapRB", &compile_options::swapRB)
    .def_readwrite("mean", &compile_options::mean)
    .def_readwrite("std", &compile_options::std)
    .def_readwrite("input_range", &compile_options::input_range)
    .def_readwrite("output_range", &compile_options::output_range)
    .def_readwrite("input_shape", &compile_options::input_shape)
    .def_readwrite("letterbox_value", &compile_options::letterbox_value)
    .def_readwrite("input_type", &compile_options::input_type)
    .def_readwrite("output_type", &compile_options::output_type)
```

```
.def_readwrite("input_layout", &compile_options::input_layout)
.def_readwrite("output_layout", &compile_options::output_layout)
.def_readwrite("model_layout", &compile_options::model_layout)
.def_readwrite("is_fpga", &compile_options::is_fpga)
.def_readwrite("dump_ir", &compile_options::dump_ir)
.def_readwrite("dump_asm", &compile_options::dump_asm)
.def_readwrite("dump_quant_error", &compile_options::dump_quant_error)
.def_readwrite("dump_dir", &compile_options::dump_dir)
.def_readwrite("benchmark_only", &compile_options::benchmark_only);
```

【属性】

名称	类型	描述
target	string	指定编译目标, 如'k210', 'k510', 'k230'
quant_type	string	指定数据量化类型, 如'uint8', 'int8'
w_quant_type	string	指定权重量化类型, 如'uint8', 'int8', 默认为'uint8'
use_mse_quant_w	bool	指定权重量化时是否使用最小化均方误差(mean-square error, MSE)算法优化量化参数
split_w_to_act	bool	指定是否将部分权重数据平衡到激活数据中
preprocess	bool	是否开启前处理, 默认为 False
swapRB	bool	是否交换 RGB 输入数据的红和蓝两个通道(RGB->BGR 或者 BGR->RGB), 默认为 False
mean	list	前处理标准化参数均值, 默认为[0, 0, 0]
std	list	前处理标准化参数方差, 默认为[1, 1, 1]
input_range	list	输入数据反量化后对应浮点数的范围, 默认为[0, 1]
output_range	list	输出定点数据前对应浮点数的范围, 默认为空
input_shape	list	指定输入数据的 shape, input_shape 的 layout 需要与 input layout 保持一致, 输入数据的 input_shape 与模型的 input shape 不一致时会进行 letterbox 操作(resize/pad 等)

名称	类型	描述
letterbox_value	float	指定前处理 letterbox 的填充值
input_type	string	指定输入数据的类型, 默认为'float32'
output_type	string	指定输出数据的类型, 如'float32', 'uint8'(仅用于指定量化情况下), 默认为'float32'
input_layout	string	指定输入数据的 layout, 如'NCHW', 'NHWC'. 若输入数据 layout 与模型本身 layout 不同, nncase 会插入 transpose 进行转换
output_layout	string	指定输出数据的 layout, 如'NCHW', 'NHWC'. 若输出数据 layout 与模型本身 layout 不同, nncase 会插入 transpose 进行转换
model_layout	string	指定模型的 layout, 默认为空, 当 tf lite 模型 layout 为 'NCHW', Onnx 和 Caffe 模型 layout 为 'NHWC' 时需指定
dump_ir	bool	指定是否 dump IR, 默认为 False
dump_asm	bool	指定是否 dump asm 汇编文件, 默认为 False
dump_quant_error	bool	指定是否 dump 量化前后的模型误差
dump_dir	bool	前面指定 dump_ir 等开关后, 这里指定 dump 的目录, 默认为空字符串

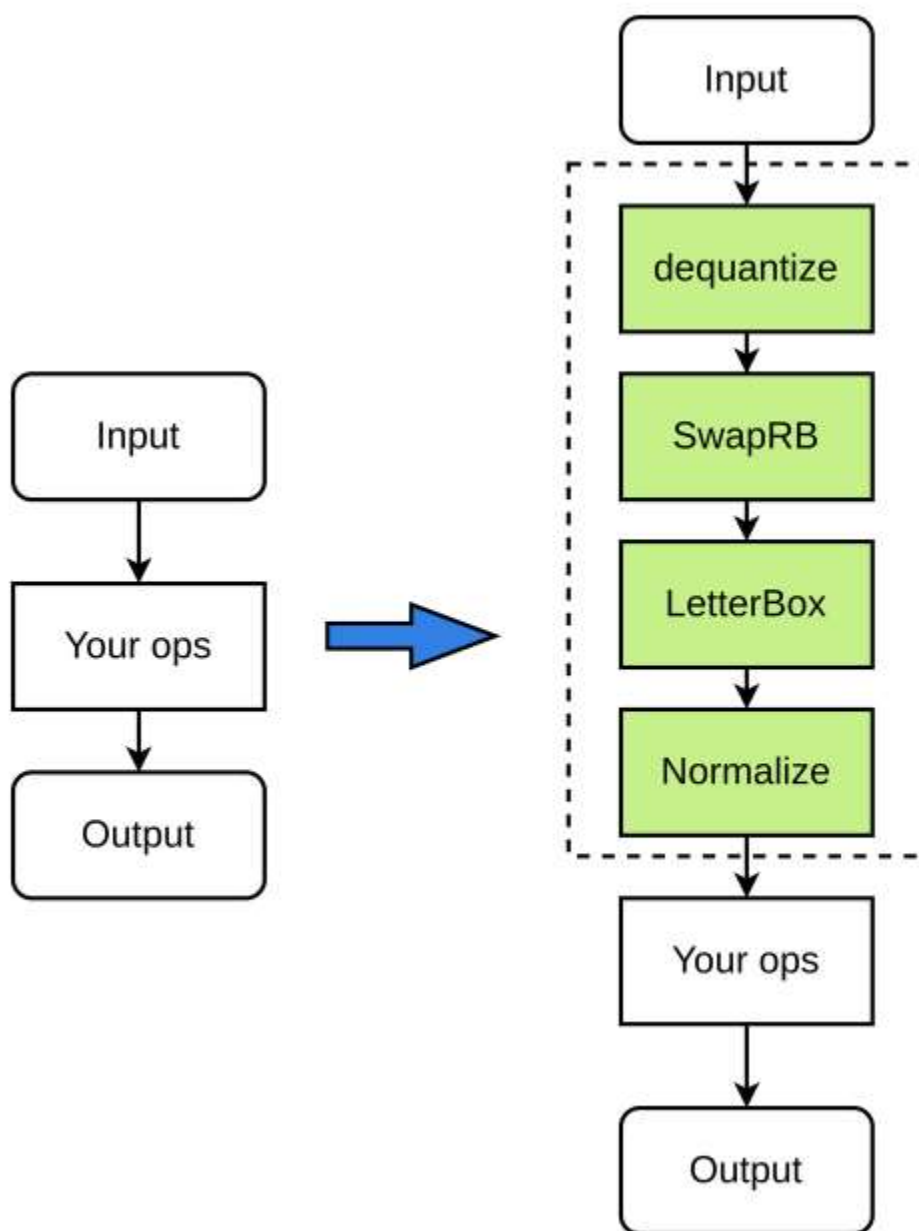
【注意】

1. input range 为浮点数的范围, 即如果输入数据类型为 uint8, 则 input range 为反量化到浮点之后的范围 (可以不为 0~1), 可以自由指定.
2. input_shape 需要按照 input_layout 进行指定, 以[1, 224, 224, 3]为例, 如果 input_layout 为 NCHW, 则 input_shape 需指定为[1,3,224,224];input_layout 为 NHWC, 则 input_shape 需指定为 [1,224,224,3];
3. mean 和 std 为浮点数进行 normalize 的参数, 用户可以自由指定;
4. 使用 letterbox 功能时, 需要限制输入 size 在 1.5MB 内, 单 channel 的 size 在 0.75MB 内;

例如:

1. 输入数据类型设定为 uint8, input_range 设定为[0,255], 则反量化的作用只是进行类型转化, 将 uint8 的数据转化为 float32, mean 和 std 参数仍然可以按照 0~255 的数据进行指定.
2. 输入数据类型设定为 uint8, input_range 设定为[0,1], 则会将定点数反量化为范围为[0,1]的浮点数, mean 和 std 需要按照新的浮点数范围进行指定.

前处理流程如下(图中绿色节点皆为可选):



【示例】

实例化 CompileOptions, 配置各属性的值

```
# compile_options
compile_options = nncase.CompileOptions()
compile_options.target = target
compile_options.input_type = 'float32' # or 'uint8' 'int8'
compile_options.output_type = 'float32' # or 'uint8' 'int8'. Only work in PTQ
compile_options.output_range = [] # Only work in PTQ and output type is not "float32"
compile_options.preprocess = True # if False, the args below will unworked
compile_options.swapRB = True
compile_options.input_shape = [1,224,224,3] # keep layout same as input layout
compile_options.input_layout = 'NHWC'
compile_options.output_layout = 'NHWC'
compile_options.model_layout = " # Specific it when tf lite model with "NCHW" layout and
Onnx(Caffe) model with "NHWC" layout
compile_options.mean = [0,0,0]
compile_options.std = [1,1,1]
compile_options.input_range = [0,1]
compile_options.letterbox_value = 114. # pad what you want
compile_options.dump_ir = True
compile_options.dump_asm = True
compile_options.dump_dir = 'tmp'
```

2.2.2 ImportOptions

【描述】

ImportOptions 类, 用于配置 nncase 导入选项

【定义】

```
py::class_<import_options>(m, "ImportOptions")
    .def(py::init())
    .def_readwrite("output_arrays", &import_options::output_arrays);
```

【属性】

名称	类型	描述
output_arrays	string	输出名称

【示例】

实例化 ImportOptions, 配置各属性的值

```
# import_options
import_options = nncase.ImportOptions()
import_options.output_arrays = 'output' # Your output node name
```

2.2.3 PTQTensorOptions

【描述】

PTQTensorOptions 类, 用于配置 nncase PTQ 选项

【定义】

```
py::class<ptq_tensor_options>(m, "PTQTensorOptions")
    .def(py::init())
    .def_readwrite("calibrate_method", &ptq_tensor_options::calibrate_method)
    .def_readwrite("samples_count", &ptq_tensor_options::samples_count)
    .def("set_tensor_data", [](ptq_tensor_options &o, py::bytes bytes) {
        uint8_t *buffer;
        py::ssize_t length;
        if (PyBytes_AsStringAndSize(bytes.ptr(), reinterpret_cast<char **>(&buffer), &length))
            throw std::invalid_argument("Invalid bytes");
        o.tensor_data.assign(buffer, buffer + length);
    });
```

【属性】

名称	类型	描述
calibrate_method	string	校准方法, 支持'no_clip', 'l2', 'kld_m0', 'kld_m1', 'kld_m2', 'cdf', 默认是'no_clip'
samples_count	Int	样本个数

【示例】

```
# ptq_options
```



```
ptq_options = nncase.PTQTensorOptions()
ptq_options.samples_count = cfg.generate_calibs.batch_size
```

2.2.4 set_tensor_data

【描述】

设置 tensor 数据

【定义】

```
set_tensor_data(calib_data)
```

【参数】

名称	类型	描述
calib_data	byte[]	读取的校准数据

【返回值】

无

【示例】

```
ptq_options.set_tensor_data(np.asarray([sample['data'] for sample in self.calibs]).tobytes())
```

2.2.5 Compiler

【描述】

Compiler 类, 用于编译神经网络模型

【定义】

```
py::class_<compiler>(m, "Compiler")
    .def(py::init(&compiler::create))
    .def("import_tflite", &compiler::import_tflite)
    .def("import_onnx", &compiler::import_onnx)
    .def("import_caffe", &compiler::import_caffe)
```

```
.def("compile", &compiler::compile)
.def("use_ptq", py::overload_cast<ptq_tensor_options>(&compiler::use_ptq))
.def("gencode", [](compiler &c, std::ostream &stream) { c.gencode(stream); })
.def("gencode_tobytes", [](compiler &c) {
    std::stringstream ss;
    c.gencode(ss);
    return py::bytes(ss.str());
})
.def("create_evaluator", [](compiler &c, uint32_t stage) {
    auto &graph = c.graph(stage);
    return std::make_unique<graph_evaluator>(c.target(), graph);
});
```

2.2.6 import_tflite

【描述】

导入 tflite 模型

【定义】

```
import_tflite(model_content, import_options)
```

【参数】

名称	类型	描述
model_content	byte[]	读取的模型内容
import_options	ImportOptions	导入选项

【返回值】

无

【示例】

```
model_content = read_model_file(model)
compiler.import_tflite(model_content, import_options)
```

2.2.7 import_onnx

【描述】

导入 onnx 模型

【定义】

```
import_onnx(model_content, import_options)
```

【参数】

名称	类型	描述
model_content	byte[]	读取的模型内容
import_options	ImportOptions	导入选项

【返回值】

无

【示例】

```
model_content = read_model_file(model)
compiler.import_onnx(model_content, import_options)
```

2.2.8 import_caffe

【描述】

导入 caffe 模型

【定义】

```
import_caffe(caffemodel, prototxt)
```

【参数】

名称	类型	描述
caffemodel	byte[]	读取的 caffemodel 内容
prototxt	byte[]	读取的 prototxt 内容

【返回值】

无

【示例】

```
# import
caffemodel = read_model_file('test.caffemodel')
prototxt = read_model_file('test.prototxt')
compiler.import_caffe(caffemodel, prototxt)
```

2.2.9 use_ptq

【描述】

设置 PTQ 配置选项.

- K230 默认必须使用量化。

【定义】

```
use_ptq(ptq_options)
```

【参数】

名称	类型	描述
ptq_options	PTQTensorOptions	PTQ 配置选项

【返回值】

无

【示例】

```
compiler.use_ptq(ptq_options)
```

2.2.10 compile

【描述】

编译神经网络模型

【定义】

```
compile()
```

【参数】

无

【返回值】

无

【示例】

```
compiler.compile()
```

2.2.11 gencode_tobytes

【描述】

生成 kmodel 字节流

【定义】

```
gencode_tobytes()
```

【参数】

无

【返回值】

bytes[]

【示例】

```
kmodel = compiler.gencode_tobytes()
with open(os.path.join(infer_dir, 'test.kmodel'), 'wb') as f:
    f.write(kmodel)
```

2.3 示例

下面示例中使用到的模型和 python 编译脚本

- 原始模型文件位于/path/to/k230_sdk/src/big/ai/examples/models 目录
- python 编译脚本位于/path/to/k230_sdk/src/big/ai/examples/scripts 目录

2.3.1 编译 tflite 模型

mbv2_tflite.py 脚本如下

```
import os
import argparse
import numpy as np
from PIL import Image
import nncase

def read_model_file(model_file):
    with open(model_file, 'rb') as f:
        model_content = f.read()
    return model_content

def generate_data(shape, batch, calib_dir):
    img_paths = [os.path.join(calib_dir, p) for p in os.listdir(calib_dir)]
    data = []
    for i in range(batch):
        assert i < len(img_paths), "calibration images not enough."
        img_data = Image.open(img_paths[i]).convert('RGB')
        img_data = img_data.resize((shape[3], shape[2]), Image.BILINEAR)
        img_data = np.asarray(img_data, dtype=np.uint8)
        img_data = np.transpose(img_data, (2, 0, 1))
```

```
        data.append(img_data)
    return np.array(data)

def main():
    parser = argparse.ArgumentParser(prog="nncase")
    parser.add_argument("--target", type=str, help='target to run')
    parser.add_argument("--model", type=str, help='model file')
    parser.add_argument("--dataset", type=str, help='calibration_dataset')
    args = parser.parse_args()

    input_shape = [1, 3, 224, 224]
    dump_dir = 'tmp/mbv2_tflite'

    # compile_options
    compile_options = nncase.CompileOptions()
    compile_options.target = args.target
    compile_options.preprocess = True
    compile_options.swapRB = False
    compile_options.input_shape = input_shape
    compile_options.input_type = 'uint8'
    compile_options.input_range = [0, 255]
    compile_options.mean = [127.5, 127.5, 127.5]
    compile_options.std = [127.5, 127.5, 127.5]
    compile_options.input_layout = 'NCHW'
    compile_options.output_layout = 'NHWC'
    compile_options.dump_ir = True
    compile_options.dump_asm = True
    compile_options.dump_dir = dump_dir
    compile_options.quant_type = 'uint8'

    # compiler
    compiler = nncase.Compiler(compile_options)

    # ptq_options
    ptq_options = nncase.PTQTensorOptions()
    ptq_options.samples_count = 6
    ptq_options.set_tensor_data(generate_data(input_shape, ptq_options.samples_count,
args.dataset).tobytes())
```

```
compiler.use_ptq(ptq_options)

# import
model_content = read_model_file(args.model)
import_options = nncase.ImportOptions()
compiler.import_tflite(model_content, import_options)

# compile
compiler.compile()

# kmodel
kmodel = compiler.gencode_tobytes()
with open(os.path.join(dump_dir, 'test.kmodel'), 'wb') as f:
    f.write(kmodel)

if __name__ == '__main__':
    main()
```

执行如下命令即可编译 mobilenetv2 的 tflite 模型, target 为 k230

```
root@c285a41a7243:/mnt/# cd src/big/ai/examples
root@c285a41a7243:/mnt/src/big/ai/examples# python ./scripts/mbv2_tflite.py --target k230 --
model models/mbv2.tflite --dataset CalibrationSet
```

2.3.2 编译 onnx 模型

针对 onnx 模型, 建议先使用 [ONNX Simplifier](#) 进行简化, 然后再使用 nncase 编译.

yolov5s_onnx.py 脚本如下

```
import os
import argparse
import numpy as np
from PIL import Image
import onnxsim
import onnx
import nncase

def parse_model_input_output(model_file):
```



```
onnx_model = onnx.load(model_file)
input_all = [node.name for node in onnx_model.graph.input]
input_initializer = [node.name for node in onnx_model.graph.initializer]
input_names = list(set(input_all) - set(input_initializer))
input_tensors = [
    node for node in onnx_model.graph.input if node.name in input_names]

# input
inputs = []
for _, e in enumerate(input_tensors):
    onnx_type = e.type.tensor_type
    input_dict = {}
    input_dict['name'] = e.name
    input_dict['dtype'] = onnx.mapping.TENSOR_TYPE_TO_NP_TYPE[onnx_type.elem_type]
    input_dict['shape'] = [(i.dim_value if i.dim_value != 0 else d) for i, d in zip(
        onnx_type.shape.dim, [1, 3, 224, 224])]
    inputs.append(input_dict)

return onnx_model, inputs

def onnx_simplify(model_file, dump_dir):
    onnx_model, inputs = parse_model_input_output(model_file)
    onnx_model = onnx.shape_inference.infer_shapes(onnx_model)
    input_shapes = {}
    for input in inputs:
        input_shapes[input['name']] = input['shape']

    onnx_model, check = onnxsim.simplify(onnx_model, input_shapes=input_shapes)
    assert check, "Simplified ONNX model could not be validated"

    model_file = os.path.join(dump_dir, 'simplified.onnx')
    onnx.save_model(onnx_model, model_file)
    return model_file

def read_model_file(model_file):
    with open(model_file, 'rb') as f:
```

```
    model_content = f.read()
    return model_content

def generate_data_random(shape, batch):
    shape[0] *= batch
    data = np.random.randint(0, 256, shape)
    return data

def generate_data(shape, batch, calib_dir):
    img_paths = [os.path.join(calib_dir, p) for p in os.listdir(calib_dir)]
    data = []
    for i in range(batch):
        assert i < len(img_paths), "calibration images not enough."
        img_data = Image.open(img_paths[i]).convert('RGB')
        img_data = img_data.resize((shape[3], shape[2]), Image.BILINEAR)
        img_data = np.asarray(img_data, dtype=np.uint8)
        img_data = np.transpose(img_data, (2, 0, 1))
        data.append(img_data)
    return np.array(data)

def main():
    parser = argparse.ArgumentParser(prog="nncase")
    parser.add_argument("--target", type=str, help='target to run')
    parser.add_argument("--model", type=str, help='model file')
    args = parser.parse_args()

    input_shape = [1, 3, 320, 320]

    dump_dir = 'tmp/yolov5s_onnx'
    if not os.path.exists(dump_dir):
        os.makedirs(dump_dir)

    # onnx simplify
    model_file = onnx_simplify(args.model, dump_dir)

    # compile_options
    compile_options = nncase.CompileOptions()
```

```
compile_options.target = args.target
compile_options.preprocess = True
compile_options.swapRB = False
compile_options.input_shape = input_shape
compile_options.input_type = 'uint8'
compile_options.input_range = [0, 255]
compile_options.mean = [0, 0, 0]
compile_options.std = [255, 255, 255]
compile_options.input_layout = 'NCHW'
compile_options.output_layout = 'NCHW'
compile_options.dump_ir = True
compile_options.dump_asm = True
compile_options.dump_dir = dump_dir
compile_options.quant_type = 'uint8'

# compiler
compiler = nncase.Compiler(compile_options)

# ptq_options
ptq_options = nncase.PTQTensorOptions()
ptq_options.samples_count = 10
ptq_options.set_tensor_data(generate_data_random(input_shape,
ptq_options.samples_count).tobytes())
compiler.use_ptq(ptq_options)

# import
model_content = read_model_file(model_file)
import_options = nncase.ImportOptions()
compiler.import_onnx(model_content, import_options)

# compile
compiler.compile()

# kmodel
kmodel = compiler.gencode_tobytes()
with open(os.path.join(dump_dir, 'test.kmodel'), 'wb') as f:
    f.write(kmodel)
```

```
if __name__ == '__main__':  
    main()
```

执行如下命令即可编译 onnx 模型, target 为 k230

```
root@c285a41a7243:/mnt/# cd src/big/ai/examples  
root@c285a41a7243: /mnt/src/big/ai/examples # python ./scripts/yolov5s_onnx.py --target k230  
--model models/yolov5s.onnx
```

2.3.3 编译 caffe 模型

conv2d_caffe.py 脚本如下

```
import os  
import argparse  
import numpy as np  
import nncase  
  
def read_model_file(model_file):  
    with open(model_file, 'rb') as f:  
        model_content = f.read()  
    return model_content  
  
def generate_data(shape, batch):  
    shape[0] *= batch  
    data = np.random.randint(0, 256, shape)  
    return data  
  
def main():  
    parser = argparse.ArgumentParser(prog="nncase")  
    parser.add_argument("--target", type=str, help='target to run')  
    parser.add_argument("--caffemodel", type=str, help='caffemodel file')  
    parser.add_argument("--prototxt", type=str, help='prototxt file')  
    args = parser.parse_args()  
  
    input_shape = [3, 3, 28, 28]  
    dump_dir = 'tmp/conv2d_caffe'
```

```
# compile_options
compile_options = nncase.CompileOptions()
compile_options.target = args.target
compile_options.preprocess = True
compile_options.swapRB = False
compile_options.input_shape = input_shape
compile_options.input_type = 'uint8'
compile_options.input_range = [0, 1]
compile_options.mean = [0, 0, 0]
compile_options.std = [1, 1, 1]
compile_options.input_layout = 'NCHW'
compile_options.output_layout = 'NCHW'
compile_options.dump_ir = True
compile_options.dump_asm = True
compile_options.dump_dir = dump_dir
compile_options.quant_type = 'uint8'

# compiler
compiler = nncase.Compiler(compile_options)

# ptq_options
ptq_options = nncase.PTQTensorOptions()
ptq_options.samples_count = 10
ptq_options.set_tensor_data(generate_data(input_shape,
ptq_options.samples_count).tobytes())
compiler.use_ptq(ptq_options)

# import
import_options = nncase.ImportOptions()
caffemodel = read_model_file(args.caffemodel)
prototxt = read_model_file(args.prototxt)
compiler.import_caffe(caffemodel, prototxt)

# compile
compiler.compile()

# kmodel
kmodel = compiler.gencode_tobytes()
```

```
with open(os.path.join(dump_dir, 'test.kmodel'), 'wb') as f:
    f.write(kmodel)

if __name__ == '__main__':
    main()
```

执行如下命令即可编译 conv2d 的 caffe 模型, target 为 k230

```
root@c285a41a7243:/mnt/# cd src/big/ai/examples
root@c285a41a7243:/mnt/src/big/ai/examples # python scripts/conv2d_caffe.py --target k230 --
caffemodel models/conv2d_caffe/test.caffemodel --prototxt models/conv2d_caffe/test.prototxt
```

3 模拟器 APIs(Python)

除了编译模型 APIs, nncase 还提供了推理模型的 APIs, 在 PC 上可推理编译模型生成的 kmodel, 用来验证 nncase 推理结果和相应深度学习框架的 runtime 的结果是否一致等.

3.1 APIs

3.1.1 MemoryRange

【描述】

MemoryRange 类, 用于表示内存范围

【定义】

```
py::class_<memory_range>(m, "MemoryRange")
    .def_readwrite("location", &memory_range::memory_location)
    .def_property(
        "dtype", [](const memory_range &range) { return to_dtype(range.datatype); },
        [](memory_range &range, py::object dtype) { range.datatype =
from_dtype(py::dtype::from_args(dtype)); })
    .def_readwrite("start", &memory_range::start)
    .def_readwrite("size", &memory_range::size);
```

【属性】

名称	类型	描述
location	int	内存位置, 0 表示 input, 1 表示 output, 2 表示 rdata, 3 表示 data, 4 表示 shared_data
dtype	python 数据类型	数据类型
start	int	内存起始地址
Size	int	内存大小

【示例】

```
mr = nncase.MemoryRange()
```

3.1.2 RuntimeTensor

【描述】

RuntimeTensor 类, 用于表示运行时 tensor

【定义】

```
py::class_<runtime_tensor>(m, "RuntimeTensor")
    .def_static("from_numpy", [](py::array arr) {
        auto src_buffer = arr.request();
        auto datatype = from_dtype(arr.dtype());
        auto tensor = host_runtime_tensor::create(
            datatype,
            to_rt_shape(src_buffer.shape),
            to_rt_strides(src_buffer.itemsize, src_buffer.strides),
            gsl::make_span(reinterpret_cast<gsl::byte*>(src_buffer.ptr), src_buffer.size *
src_buffer.itemsize),
            [=](gsl::byte *) { arr.dec_ref(); })
            .unwrap_or_throw();
        arr.inc_ref();
        return tensor;
    })
    .def("copy_to", [](runtime_tensor &from, runtime_tensor &to) {
        from.copy_to(to).unwrap_or_throw();
    });
```

```

})
.def("to_numpy", [](runtime_tensor &tensor) {
    auto host = tensor.as_host().unwrap_or_throw();
    auto src_map = std::move(hrt::map(host, hrt::map_read).unwrap_or_throw());
    auto src_buffer = src_map.buffer();
    return py::array(
        to_dtype(tensor.datatype()),
        tensor.shape(),
        to_py_strides(runtime::get_bytes(tensor.datatype()), tensor.strides()),
        src_buffer.data());
})
.def_property_readonly("dtype", [](runtime_tensor &tensor) {
    return to_dtype(tensor.datatype());
})
.def_property_readonly("shape", [](runtime_tensor &tensor) {
    return to_py_shape(tensor.shape());
});

```

【属性】

名称	类型	描述
dtype	python 数据类型	Tensor 的数据类型
shape	list	tensor 的形状

3.1.3 from_numpy

【描述】

从 numpy.ndarray 构造 RuntimeTensor 对象

【定义】

```
from_numpy(py::array arr)
```

【参数】

名称	类型	描述
Arr	numpy.ndarray	numpy.ndarray 对象

【返回值】

RuntimeTensor

【示例】

```
tensor = nncase.RuntimeTensor.from_numpy(self.inputs[i]['data'])
```

3.1.4 copy_to

【描述】

拷贝 RuntimeTensor

【定义】

```
copy_to(RuntimeTensor to)
```

【参数】

名称	类型	描述
to	RuntimeTensor	RuntimeTensor 对象

【返回值】

无

【示例】

```
sim.get_output_tensor(i).copy_to(to)
```

3.1.5 to_numpy

【描述】

将 RuntimeTensor 转换为 numpy.ndarray 对象

【定义】

```
to_numpy()
```

【参数】

无

【返回值】

numpy.ndarray 对象

【示例】

```
arr = sim.get_output_tensor(i).to_numpy()
```

3.1.6 Simulator

【描述】

Simulator 类, 用于在 PC 上推理 kmodel

【定义】

```
py::class_<interpreter>(m, "Simulator")
    .def(py::init())
    .def("load_model", [](interpreter &interp, gsl::span<const gsl::byte> buffer)
    { interp.load_model(buffer).unwrap_or_throw(); })
    .def_property_readonly("inputs_size", &interpreter::inputs_size)
    .def_property_readonly("outputs_size", &interpreter::outputs_size)
    .def("get_input_desc", &interpreter::input_desc)
    .def("get_output_desc", &interpreter::output_desc)
    .def("get_input_tensor", [](interpreter &interp, size_t index) { return
    interp.input_tensor(index).unwrap_or_throw(); })
    .def("set_input_tensor", [](interpreter &interp, size_t index, runtime_tensor tensor) { return
    interp.input_tensor(index, tensor).unwrap_or_throw(); })
    .def("get_output_tensor", [](interpreter &interp, size_t index) { return
    interp.output_tensor(index).unwrap_or_throw(); })
```

```
.def("set_output_tensor", [](interpreter &interp, size_t index, runtime_tensor tensor) { return  
interp.output_tensor(index, tensor).unwrap_or_throw(); })  
.def("run", [](interpreter &interp) { interp.run().unwrap_or_throw(); });
```

【属性】

名称	类型	描述
inputs_size	int	输入个数
outputs_size	int	输出个数

【示例】

```
sim = nncase.Simulator()
```

3.1.7 load_model

【描述】

加载 kmodel

【定义】

```
load_model(model_content)
```

【参数】

名称	类型	描述
model_content	byte[]	kmodel 字节流

【返回值】

无

【示例】

```
sim.load_model(kmodel)
```

3.1.8 get_input_desc

【描述】

获取指定索引的输入的描述信息

【定义】

```
get_input_desc(index)
```

【参数】

名称	类型	描述
index	int	输入的索引

【返回值】

MemoryRange

【示例】

```
input_desc_0 = sim.get_input_desc(0)
```

3.1.9 get_output_desc

【描述】

获取指定索引的输出的描述信息

【定义】

```
get_output_desc(index)
```

【参数】

名称	类型	描述
index	int	输出的索引

【返回值】

MemoryRange

【示例】

```
output_desc_0 = sim.get_output_desc(0)
```

3.1.10 get_input_tensor

【描述】

获取指定索引的输入的 RuntimeTensor

【定义】

```
get_input_tensor(index)
```

【参数】

名称	类型	描述
index	int	输入 tensor 的索引

【返回值】

RuntimeTensor

【示例】

```
input_tensor_0 = sim.get_input_tensor(0)
```

3.1.11 set_input_tensor

【描述】

设置指定索引的输入的 RuntimeTensor

【定义】

```
set_input_tensor(index, tensor)
```

【参数】

名称	类型	描述
index	int	输入 tensor 的索引
tensor	RuntimeTensor	输入 tensor

【返回值】

无

【示例】

```
sim.set_input_tensor(0, nncase.RuntimeTensor.from_numpy(self.inputs[0]['data']))
```

3.1.12 get_output_tensor

【描述】

获取指定索引的输出的 RuntimeTensor

【定义】

```
get_output_tensor(index)
```

【参数】

名称	类型	描述
index	int	输出 tensor 的索引

【返回值】

RuntimeTensor

【示例】

```
output_arr_0 = sim.get_output_tensor(0).to_numpy()
```

3.1.13 set_output_tensor

【描述】

设置指定索引的输出的 RuntimeTensor

【定义】

```
set_output_tensor(index, tensor)
```

【参数】

名称	类型	描述
index	int	输出 tensor 的索引
tensor	RuntimeTensor	输出 tensor

【返回值】

无

【示例】

```
sim.set_output_tensor(0, tensor)
```

3.1.14 run

【描述】

运行 kmodel 推理

【定义】

```
run()
```

【参数】

无

【返回值】

无

【示例】

```
sim.run()
```

3.2 示例

前置条件: yolov5s_onnx.py 脚本已编译过 yolov5s.onnx 模型

yolov5s_onnx_simu.py 位于/path/to/k230_sdk/src/big/ai/examples/scripts 子目录， 内容如下

```
import os
import copy
import argparse
import numpy as np
import onnx
import onnxruntime as ort
import nncase

def read_model_file(model_file):
    with open(model_file, 'rb') as f:
        model_content = f.read()
    return model_content

def cosine(gt, pred):
    return (gt @ pred) / (np.linalg.norm(gt, 2) * np.linalg.norm(pred, 2))

def main():
    parser = argparse.ArgumentParser(prog="nncase")
    parser.add_argument("--model", type=str, help='original model file')
    parser.add_argument("--model_input", type=str, help='input bin file for original model')
    parser.add_argument("--kmodel", type=str, help='kmodel file')
    parser.add_argument("--kmodel_input", type=str, help='input bin file for kmodel')
    args = parser.parse_args()

    # cpu inference
    ort_session = ort.InferenceSession(args.model)
```



```
output_names = []
model_outputs = ort_session.get_outputs()
for i in range(len(model_outputs)):
    output_names.append(model_outputs[i].name)
model_input = ort_session.get_inputs()[0]
model_input_name = model_input.name
model_input_type = np.float32
model_input_shape = model_input.shape
model_input_data = np.fromfile(args.model_input,
model_input_type).reshape(model_input_shape)
cpu_results = []
cpu_results = ort_session.run(output_names, { model_input_name : model_input_data })

# create simulator
sim = nncase.Simulator()

# read kmodel
kmodel = read_model_file(args.kmodel)

# load kmodel
sim.load_model(kmodel)

# read input.bin
input_tensor=sim.get_input_tensor(0).to_numpy()
input = np.fromfile(args.kmodel_input, input_tensor.dtype).reshape(input_tensor.shape)

# set input for simulator
sim.set_input_tensor(0, nncase.RuntimeTensor.from_numpy(input))

# simulator inference
nncase_results = []
sim.run()
for i in range(sim.outputs_size):
    nncase_result = sim.get_output_tensor(i).to_numpy()
    nncase_results.append(copy.deepcopy(nncase_result))

# compare
for i in range(sim.outputs_size):
```

```
cos = cosine(np.reshape(nncase_results[i], (-1)), np.reshape(cpu_results[i], (-1)))
print('output {0} cosine similarity : {1}'.format(i, cos))

if __name__ == '__main__':
    main()
```

执行推理脚本

```
root@5f718e19f8a7:/mnt/# cd src/big/ai/examples
root@5f718e19f8a7:/mnt/src/big/ai/examples# python scripts/yolov5s_onnx_simu.py --model
models/yolov5s.onnx --model_input object_detect/data/input_fp32.bin --kmodel
tmp/yolov5s_onnx/test.kmodel --kmodel_input object_detect/data/input_uint8.bin
```

nncase simulator 和 cpu 推理结果对比如下

```
... ..
[error] Mmu Item = 0 can only be config by once!
output 0 cosine similarity : 0.9998742341995239
output 1 cosine similarity : 0.999858021736145
output 2 cosine similarity : 0.9998511672019958
```

4 KPU 运行时 APIs(C++)

4.1 简介

KPU 运行时 APIs 用于在 AI 设备加载 kmodel，设置输入数据，执行 kpu/cpu 计算，获取输出数据等。

目前只提供 C++ APIs, 相关的头文件和静态库在 nncase sdk/riscv64 目录下。

```
$ tree -L 3 riscv64/
riscv64/
├── gsl
│   └── gsl-lite.hpp
├── gsl-lite
│   └── gsl-lite.hpp
```

```
├── mpark
│   ├── config.hpp
│   ├── in_place.hpp
│   ├── lib.hpp
│   └── variant.hpp
└── nncase
    ├── include
    │   └── nncase
    └── lib
        ├── cmake
        ├── libnncase.functional.a
        ├── libnncase.rt_modules.k230.a
        └── libnncase.runtime.a
```

8 directories, 9 files

4.2 APIs

4.2.1 hrt::create

【描述】

创建 runtime_tensor

【定义】

(1) NNCASE_API result<runtime_tensor> create(datatype_t datatype, runtime_shape_t shape, memory_pool_t pool = pool_cpu_only, uintptr_t physical_address = 0) noexcept;

(2) NNCASE_API result<runtime_tensor> create(datatype_t datatype, runtime_shape_t shape, gsl::span<gsl::byte> data, bool copy, memory_pool_t pool = pool_shared, uintptr_t physical_address = 0) noexcept;

【参数】

名称	类型	描述
datatype	datatype_t	数据类型, 如 dt_float32, dt_uint8 等
shape	runtime_shape_t	tensor 的形状
data	gsl::span<gsl::byte>	用户态数据 buffer
pool	memory_pool_t	内存池类型, 默认值为 pool_shared
physical_address	uintptr_t	物理地址, 默认值为 0. 非 0 表示用户指定物理地址

【返回值】

result<runtime_tensor>

【示例】

```
// create input
auto in_shape = interp.input_shape(0);
auto input_tensor = host_runtime_tensor::create(dt_float32, in_shape,
                                                {(gsl::byte *)mat.data, mat.cols * mat.rows * mat.elemSize()},
                                                true, hrt::pool_shared).expect("cannot create input tensor");
```

4.2.2 hrt::sync

【描述】

同步 tensor 的 cache。

- 对用户的输入数据, 需要调用 此接口的 sync_write_back 确保数据已刷入 ddr.
- 对 gnne/ai2d 计算后输出数据, 默认 gnne/ai2d runtime 已做了 sync_invalidate 处理。

【定义】

```
NNCASE_API result<void> sync(runtime_tensor &tensor, sync_op_t op, bool force = false) noexcept;
```

【参数】

名称	类型	描述
tensor	runtime_tensor	要操作的 tensor
op	sync_op_t	sync_invalidate(将 tensor 的 cache invalidate)或 sync_write_back(将 tensor 的 cache 写入 ddr)
force	bool	是否强制执行

【返回值】

result<void>

【示例】

```
// 2. set inputs
for (size_t i = 2, j = 0; i < 2 + interp.inputs_size(); i++, j++)
{
    auto desc = interp.input_desc(j);
    auto shape = interp.input_shape(j);
    auto tensor = host_runtime_tensor::create(desc.datatype, shape,
hrt::pool_shared).expect("cannot create input tensor");
    auto mapped_buf = std::move(hrt::map(tensor, hrt::map_write).unwrap());
    read_binary_file(argv[i], reinterpret_cast<char *>(mapped_buf.buffer().data()));
    auto ret = mapped_buf.unmap();
    ret = hrt::sync(tensor, hrt::sync_write_back, true);
    if (!ret.is_ok())
    {
        std::cerr << "hrt::sync failed" << std::endl;
        std::abort();
    }
    interp.input_tensor(j, tensor).expect("cannot set input tensor");
}
```

4.2.3 interpreter::load_model

【描述】

加载 kmodel 模型

【定义】

```
NNCASE_NODISCARD result<void> load_model(gsl::span<const gsl::byte> buffer) noexcept;
```

【参数】

名称	类型	描述
buffer	<code>gsl::span <const gsl::byte></code>	kmodel buffer

【返回值】

result<void>

【示例】

```
interpreter interp;  
auto model = read_binary_file<unsigned char>(kmodel);  
interp.load_model(((const gsl::byte *)model.data(), model.size())).expect("cannot load model.");
```

4.2.4 interpreter::inputs_size

【描述】

获取模型输入的个数

【定义】

```
size_t inputs_size() const noexcept;
```

【参数】

无

【返回值】

size_t

【示例】

```
auto inputs_size = interp.inputs_size();
```

4.2.5 interpreter::outputs_size

【描述】

获取模型输出的个数

【定义】

```
size_t outputs_size() const noexcept;
```

【参数】

无

【返回值】

size_t

【示例】

```
auto outputs_size = interp.outputs_size();
```

4.2.6 interpreter::input_shape

【描述】

获取模型指定输入的形状

【定义】

```
const runtime_shape_t &input_shape(size_t index) const noexcept;
```

【参数】

名称	类型	描述
index	size_t	输入的索引

【返回值】

runtime_shape_t

【示例】

```
auto shape = interp.input_shape(0);
```

4.2.7 interpreter::output_shape

【描述】

获取模型指定输出的形状

【定义】

```
const runtime_shape_t &output_shape(size_t index) const noexcept;
```

【参数】

名称	类型	描述
index	size_t	输出的索引

【返回值】

runtime_shape_t

【示例】

```
auto shape = interp.output_shape(0);
```

4.2.8 interpreter::input_tensor

【描述】

获取/设置指定索引的输入 tensor

【定义】

```
(1) result<runtime_tensor> input_tensor(size_t index) noexcept;
```



```
(2) result<void> input_tensor(size_t index, runtime_tensor tensor) noexcept;
```

【参数】

名称	类型	描述
index	size_t	输入的索引
tensor	runtime_tensor	输入对应的 runtime tensor

【返回值】

(1) 返回 result<runtime_tensor>

(2) 返回 result <void>

【示例】

```
// set input
interp.input_tensor(0, input_tensor).expect("cannot set input tensor");
```

4.2.9 interpreter:: output_tensor

【描述】

获取/设置指定索引的输出 tensor

【定义】

```
(1) result<runtime_tensor> output_tensor(size_t index) noexcept;
(2) result<void> output_tensor(size_t index, runtime_tensor tensor) noexcept;
```

【参数】

名称	类型	描述
index	size_t	输出的索引
tensor	runtime_tensor	输出对应的 runtime tensor

【返回值】

(1) 返回 result<runtime_tensor>

(2) 返回 result <void>

【示例】

```
// get output
auto output_tensor = interp.output_tensor(0).expect("cannot get output tensor");
```

4.2.10 interpreter::run

【描述】

执行 kpu 计算

【定义】

```
result<void> run() noexcept;
```

【参数】

无

【返回值】

返回 result <void>

【示例】

```
// run
interp.run().expect("error occurred in running model");
```

4.3 示例

```
// load kmodel
interpreter interp;
auto kmodel = read_binary_file<unsigned char>(kmodel_file);
```

```
interp.load_model({ (const gsl::byte *)kmodel.data(), kmodel.size() }).expect("cannot load model.");

// create input tensor
auto input_desc = interp.input_desc(0);
auto input_shape = interp.input_shape(0);
auto input_tensor = host_runtime_tensor::create(input_desc.datatype, input_shape, hrt::pool_shared).expect("cannot create input tensor");
interp.input_tensor(0, input_tensor).expect("cannot set input tensor");

// create output tensor
auto output_desc = interp.output_desc(0);
auto output_shape = interp.output_shape(0);
auto output_tensor = host_runtime_tensor::create(output_desc.datatype, output_shape, hrt::pool_shared).expect("cannot create output tensor");
interp.output_tensor(0, output_tensor).expect("cannot set output tensor");

// set input data
auto input_mapped_buf = std::move(hrt::map(input_tensor, hrt::map_write).unwrap());
#if USE_OPENCV
    cv::Mat img = cv::imread(image_file);
    cv::resize(img, img, cv::Size(INPUT_WIDTH, INPUT_HEIGHT), cv::INTER_NEAREST);
    auto input_vec = hwc2chw(img);
    memcpy(reinterpret_cast<char *>(input_mapped_buf.buffer().data()), input_vec.data(), input_vec.size());
#else
    read_binary_file(image_file, reinterpret_cast<char *>(input_mapped_buf.buffer().data()));
#endif
auto ret = input_mapped_buf.unmap();
hrt::sync(input_tensor, hrt::sync_write_back, true).expect("sync write_back failed");

// run
size_t counter = 1;
auto start = std::chrono::steady_clock::now();
for (size_t c = 0; c < counter; c++)
{
    interp.run().expect("error occurred in running model");
}
```

```
auto stop = std::chrono::steady_clock::now();
double duration = std::chrono::duration<double, std::milli>(stop - start).count();
std::cout << "interp.run() took: " << duration / counter << " ms" << std::endl;

// get output data
auto output_mapped_buf = std::move(hrt::map(output_tensor, hrt::map_read).unwrap());
float *output_data = reinterpret_cast<float *>(output_mapped_buf.buffer().data());
auto out_shape = interp.output_shape(0);
auto size = compute_size(out_shape);

// postprogress softmax by cpu
std::vector<float> softmax_vec(size, 0);
auto buf = softmax_vec.data();
softmax(output_data, buf, size);
auto it = std::max_element(buf, buf + size);
size_t idx = it - buf;

// load label
auto labels = read_txt_file(label_file);
std::cout << "image classify result: " << labels[idx] << "(" << *it << ")" << std::endl;
```

5 AI2D 运行时 APIs(C++)

5.1 简介

AI2D 运行时 APIs 用于在 AI 设备配置 AI2D 的参数，生成相关寄存器配置，执行 AI2D 计算等。

5.2 APIs

5.2.1 ai2d_format

【描述】

ai2d_format 用于配置输入输出的可选数据格式。

【定义】

```
enum class ai2d_format
{
    YUV420_NV12 = 0,
    YUV420_NV21 = 1,
    YUV420_I420 = 2,
    NCHW_FMT = 3,
    RGB_packed = 4,
    RAW16 = 5,
}
```

5.2.2 ai2d_interp_method

【描述】

ai2d_interp_method 用于配置可选的插值方式.

【定义】

```
enum class ai2d_interp_method
{
    tf_nearest = 0,
    tf_bilinear = 1,
    cv2_nearest = 2,
    cv2_bilinear = 3,
}
```

5.2.3 ai2d_interp_mode

【描述】

ai2d_interp_mode 用于配置可选的插值模式.

【定义】

```
enum class ai2d_interp_mode
{
    none = 0,
    align_corner = 1,
    half_pixel = 2,
```

```
}
```

5.2.4 ai2d_pad_mode

【描述】

ai2d_pad_mode 用于配置可选的 padding 模式，目前只支持常数 padding.

【定义】

```
enum class ai2d_pad_mode
{
    constant = 0,
    copy = 1,
    mirror = 2,
}
```

5.2.5 ai2d_datatype_t

【描述】

ai2d_datatype_t 用于设置 AI2D 计算过程中的数据类型.

【定义】

```
struct ai2d_datatype_t
{
    ai2d_format src_format;
    ai2d_format dst_format;
    datatype_t src_type;
    datatype_t dst_type;
    ai2d_data_loc src_loc = ai2d_data_loc::ddr;
    ai2d_data_loc dst_loc = ai2d_data_loc::ddr;
}
```

【参数】

名称	类型	描述
src_format	ai2d_format	输入数据格式
dst_format	ai2d_format	输出数据格式
src_type	datatype_t	输入数据类型
dst_type	datatype_t	输出数据类型
src_loc	ai2d_data_loc	输入数据位置，默认 ddr
dst_loc	ai2d_data_loc	输出数据位置，默认 ddr

【示例】

```
ai2d_datatype_t ai2d_dtype { ai2d_format::RAW16, ai2d_format::NCHW_FMT, datatype_t::dt_uint16,  
datatype_t::dt_uint8 };
```

5.2.6 ai2d_crop_param_t

【描述】

ai2d_crop_param_t 用于配置 crop 相关的参数.

【定义】

```
struct ai2d_crop_param_t  
{  
    bool crop_flag = false;  
    int32_t start_x = 0;  
    int32_t start_y = 0;  
    int32_t width = 0;  
    int32_t height = 0;  
}
```

【参数】

名称	类型	描述
crop_flag	bool	是否开启 crop 功能
start_x	int	宽度方向的起始像素
start_y	int	高度方向的起始像素
width	int	宽度方向的 crop 长度
height	int	高度方向的 crop 长度

【示例】

```
ai2d_crop_param_t crop_param { true, 40, 30, 400, 600 };
```

5.2.7 ai2d_shift_param_t

【描述】

ai2d_shift_param_t 用于配置 shift 相关的参数.

【定义】

```
struct ai2d_shift_param_t
{
    bool shift_flag = false;
    int32_t shift_val = 0;
}
```

【参数】

名称	类型	描述
shift_flag	bool	是否开启 shift 功能
shift_val	int	右移的比特数

【示例】


```
ai2d_shift_param_t shift_param { true, 2 };
```

5.2.8 ai2d_pad_param_t

【描述】

ai2d_pad_param_t 用于配置 pad 相关的参数.

【定义】

```
struct ai2d_pad_param_t
{
    bool pad_flag = false;
    runtime_paddings_t paddings;
    ai2d_pad_mode pad_mode = ai2d_pad_mode::constant;
    std::vector<int32_t> pad_val; // by channel
}
```

【参数】

名称	类型	描述
pad_flag	bool	是否开启 pad 功能
paddings	runtime_paddings_t	各个维度的 padding, shape=[4, 2], 分别表示 dim0 到 dim4 的前后 padding 的个数, 其中 dim0/dim1 固定配置{0, 0}
pad_mode	ai2d_pad_mode	padding 模式, 只支持 constant padding
pad_val	std::vector<int32_t>	每个 channel 的 padding value

【示例】

```
ai2d_pad_param_t pad_param { false, { { 0, 0 }, { 0, 0 }, { 0, 0 }, { 60, 60 } }, ai2d_pad_mode::constant, { 255 } };
```

5.2.9 ai2d_resize_param_t

【描述】

ai2d_resize_param_t 用于配置 resize 相关的参数.

【定义】

```
struct ai2d_resize_param_t
{
    bool resize_flag = false;
    ai2d_interp_method interp_method = ai2d_interp_method::tf_bilinear;
    ai2d_interp_mode interp_mode = ai2d_interp_mode::none;
}
```

【参数】

名称	类型	描述
resize_flag	bool	是否开启 resize 功能
interp_method	ai2d_interp_method	resize 插值方法
interp_mode	ai2d_interp_mode	resize 模式

【示例】

```
ai2d_resize_param_t    resize_param    {    true,    ai2d_interp_method::tf_bilinear,
ai2d_interp_mode::half_pixel };
```

5.2.10 ai2d_affine_param_t

【描述】

ai2d_affine_param_t 用于配置 affine 相关的参数.

【定义】

```
struct ai2d_affine_param_t
{
    bool affine_flag = false;
    ai2d_interp_method interp_method = ai2d_interp_method::cv2_bilinear;
    uint32_t cord_round = 0;
    uint32_t bound_ind = 0;
```

```
int32_t bound_val = 0;
uint32_t bound_smooth = 0;
std::vector<float> M;
}
```

【参数】

名称	类型	描述
affine_flag	bool	是否开启 affine 功能
interp_method	ai2d_interp_method	Affine 采用的插值方法
cord_round	uint32_t	整数边界 0 或者 1
bound_ind	uint32_t	边界像素模式 0 或者 1
bound_val	uint32_t	边界填充值
bound_smooth	uint32_t	边界平滑 0 或者 1
M	std::vector<float>	仿射变换矩阵对应的 vector，仿射变换为 $Y=[a_0, a_1; a_2, a_3]X+[b_0, b_1]$ ，则 $M=\{a_0,a_1,b_0,a_2,a_3,b_1\}$

【示例】

```
ai2d_affine_param_t affine_param { true, ai2d_interp_method::cv2_bilinear, 0, 0, 127, 1,
\
{ 0.5, 0.1, 0.0, 0.1, 0.5, 0.0 } };
```

5.2.11 ai2d_builder::ai2d_builder

【描述】

ai2d_builder 的构造函数.

【定义】

```
ai2d_builder(runtime::runtime_tensor &input, runtime::runtime_tensor &output, ai2d_datatype_t
ai2d_dtype,
```

```
ai2d_crop_param_t crop_param, ai2d_shift_param_t shift_param, ai2d_pad_param_t
pad_param,
ai2d_resize_param_t resize_param, ai2d_affine_param_t affine_param)
```

【参数】

名称	类型	描述
input	runtime_tensor	输入 runtime tensor
output	runtime_tensor	输出 runtime tensor
ai2d_dtype	ai2d_datatype_t	ai2d 数据类型
crop_param	ai2d_crop_param_t	crop 相关参数
shift_param	ai2d_shift_param_t	shift 相关参数
pad_param	ai2d_pad_param_t	pad 相关参数
resize_param	ai2d_resize_param_t	resize 相关参数
affine_param	ai2d_affine_param_t	affine 相关参数

【返回值】

无

【示例】

```
ai2d_builder builder { in_tensor, out_tensor, ai2d_dtype, crop_param, shift_param, pad_param,
resize_param, affine_param };
```

5.2.12 ai2d_builder:: build_schedule

【描述】

生成 AI2D 计算需要的参数.

【定义】

```
result<void> build_schedule();
```

【参数】

无

【返回值】

result<void>

【示例】

```
builder.build_schedule();
```

5.2.13 ai2d_builder:: invoke

【描述】

配置寄存器并启动 AI2D 的计算.

【定义】

```
result<void> invoke();
```

【参数】

无

【返回值】

result<void>

【示例】

```
builder.invoke().expect("error occurred in ai2d running");
```

5.3 示例

```
static void test_pad_mini_test(const char *gmodel_file, const char *expect_file)
```

```
{
    // input tensor
    runtime_shape_t in_shape { 1, 100, 150, 3 };
    auto in_tensor = host_runtime_tensor::create(datatype_t::dt_uint8, in_shape,
hrt::pool_shared).expect("cannot create input tensor");
    auto mapped_in_buf = std::move(hrt::map(in_tensor, hrt::map_write).unwrap());
    read_binary_file(gmodel_file, reinterpret_cast<char*>(mapped_in_buf.buffer().data()));
    mapped_in_buf.unmap().expect("unmap input tensor failed");
    hrt::sync(in_tensor, hrt::sync_write_back, true).expect("write back input failed");

    // output tensor
    runtime_shape_t out_shape { 1, 100, 160, 3 };
    auto out_tensor = host_runtime_tensor::create(datatype_t::dt_uint8, out_shape,
hrt::pool_shared).expect("cannot create output tensor");

    // config ai2d
    ai2d_datatype_t ai2d_dtype { ai2d_format::RGB_packed, ai2d_format::RGB_packed,
datatype_t::dt_uint8, datatype_t::dt_uint8 };
    ai2d_crop_param_t crop_param { false, 0, 0, 0, 0 };
    ai2d_shift_param_t shift_param { false, 0 };
    ai2d_pad_param_t pad_param { true, { { 0, 0 }, { 0, 0 }, { 0, 0 }, { 10, 0 } }, ai2d_pad_mode::constant,
{ 255, 10, 5 } };
    ai2d_resize_param_t resize_param { false, ai2d_interp_method::tf_bilinear,
ai2d_interp_mode::half_pixel };
    ai2d_affine_param_t affine_param { false };

    // run
    ai2d_builder builder { in_tensor, out_tensor, ai2d_dtype, crop_param, shift_param, pad_param,
resize_param, affine_param };
    auto start = std::chrono::steady_clock::now();
    builder.build_schedule().expect("error occurred in ai2d build_schedule");
    builder.invoke().expect("error occurred in ai2d invoke");
    auto stop = std::chrono::steady_clock::now();
    double duration = std::chrono::duration<double, std::milli>(stop - start).count();
    std::cout << "ai2d run: duration = " << duration << " ms, fps = " << 1000 / duration << std::endl;
}
```

5.4 注意事项

1. Affine 和 Resize 是功能是互斥的，不能同时开启
2. Shift 功能的输入格式只能是 Raw16
3. Pad value 是按通道配置的，对应的 list 元素个数要与 channel 数相等

6 FAQ

6.1 安装 wheel 包时报错: "xxx.whl is not a supported wheel on this platform."

Q: 安装 nncase wheel 包，出现 ERROR: nncase-1.0.0.20210830-cp37-cp37m-manylinux_2_24_x86_64.whl is not a supported wheel on this platform.

A: 升级 pip >= 20.3

```
$ sudo pip install --upgrade pip
```

6.2 编译模型时报错"python: symbol lookup error"

Q: 编译模型报错 "python: symbol lookup error: /usr/local/lib/python3.8/dist-packages/libnncase.modules.k230.so: undefined symbol: _ZN6nncase2ir5graph14split_subgraphESt4spanIKPNS0_4nodeELm18446744073709551615EEb"

A: 查看 nncase 和 nncase-k230 wheel 包版本是否匹配

```
root@a829814d14b7:/mnt/examples# pip list
Package Version
-----
gmssl    3.2.1
nncase   1.8.0.20220929
nncase-k230 1.9.0.20230403
numpy    1.24.2
Pillow   9.4.0
pip      23.0
pycryptodome 3.17
setuptools 45.2.0
```

wheel	0.34.2
-------	--------

6.3 上板运行 App 推理程序时报错"std::bad_alloc"

Q: 上板运行 App 推理程序, 抛出"std::bad_alloc"异常

```
$ ./cpp.sh
case ./yolov3_bfloat16 build at Sep 16 2021 18:12:03
terminate called after throwing an instance of 'std::bad_alloc'
what(): std::bad_alloc
```

A: std::bad_alloc 异常通常是因为内存分配失败导致的, 可做如下排查.

- 检查生成的 kmodel 是否超过当前系统可用内存
- 检查 App 是否存在内存泄露

6.4 上板运行 App 推理程序时报错 "data.size_bytes() == size = false (bool)"

Q: 运行 App 推理程序, 抛出"[..t_runtime_tensor.cpp:310 (create)] data.size_bytes() == size = false (bool)"异常

A: 检查设置的输入 tensor 信息, 重点是输入 shape 和每个元素占用的字节数(fp32/uint8)是否和模型一致