CVITEK TPU开发指南

> 文档版本: 1.5.8
>
> 发布日期: 2022-09-29
>
> 适用于CV183x/CV182x/CV181x系列芯片

法律声明

本数据手册包含北京晶视智能科技有限公司（下称"晶视智能"）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。

本数据手册和本文件所含的所有信息均按"原样"提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

目 录

第1章 概述

1.1 TPU介绍

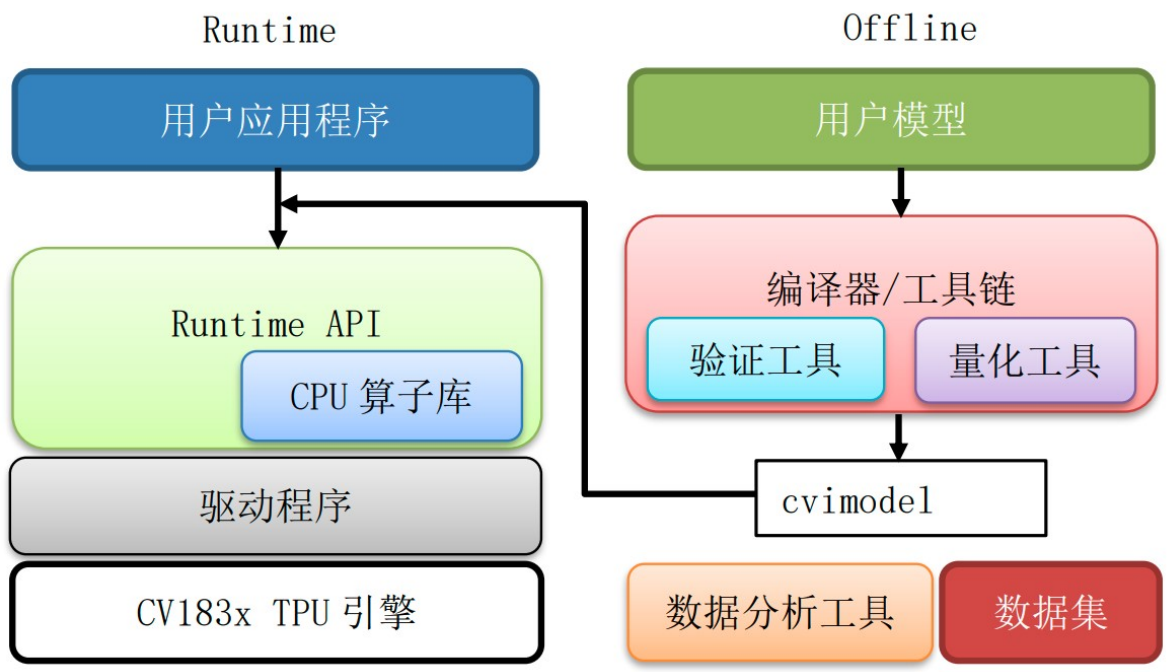
> CVITEK TPU是晶视智能开发的边缘计算SoC平台(CV183x/CV182x/CV181x)的异构计算引擎。实现了对主流神经网络运算的高效硬件加速，兼顾执行效率和可编程灵活性。计算精度方面同时支持高效的INT8和高动态范围的BF16两种模式，并通过软件平台支持灵活的混合精度配置。

1.2 工具链介绍

> CVITEK TPU工具链是一个高效，开放，透明，可扩展，模型移植全流程可控的神经网络编译优化开发工具集。

1.3 软件框架

TPU软件开发框图如下图所示:



> 软件框架由Offline工具链和Runtime模型推理库两部分组成。Offline工具链包括模型转换，编译器，量化工具等组件，完成从用户模型导入，变换，量化，优化，到代码生成等步骤，最终组装为cvimodel格式的推理模型文件。Runtime模型推理库加载cvimodel，读取运行时信息进行设置和资源分配，加载权重数据和指令序列，驱动硬件执行其中的指令序列，完成神经网络推理计算任务，输出推理结果数据。Runtime包含完整仿真平台的支持，客户可以先在仿真平台完成模型移植，验证和精度测试，再加载到真实硬件上验证和执行。

1.4 神经网络编译器

> 神经网络编译器基于MLIR框架开发。编译器完成从一个现有框架模型文件到TPU指令集的转换。具体包括下述几个工具:

- `model_transform.py`：用于将caffe/onnx框架生成的模型转换为以mlir为前端的fp32模型文件，用于后续的量化，优化和指令生成。
- `model_deploy.py`：用于将fp32 mlir文件转换成cvimodel文件
- `run_calibration.py`：对开发者指定的校准数据集执行推理计算，对每个tensor的数据进行统计，形成模型量化所需的参数。使用python进行数据读取，预处理，后处理和统计分析。
- `run_mix_precision.py`：通过校准数据集，进行混合量化，得到bf16量化表
- 仿真库：仿真库不同于mlir-interpreter，仿真库模拟真实TPU硬件的指令执行和运算逻辑。对于INT8量化计算，mlir-interpreter的计算结果与仿真库的计算结果bit-accurate一致。对于浮点类型组合计算，mlir-interpreter会和硬件有一定差异，仿真器则仍然保证结果与硬件的bit-accurate一致性。
- 数据分析工具：一组python工具集，用于对模型移植和编译过程中所产生的数据进行比较，统计，分析和问题定位。支持对转换的各个步骤输出的每层Tensor数据进行多种相似度比对，以确保模型INT8定点以及BF16推理精度达到要求。

1.5 模型转储cvimodel文件

> cvimodel文件是离线编译的最终结果，交付给runtime进行解析和在线推理。cvimodel还支持：

- 多batch和多分辨率：对于不同的batch_size和输入分辨率，由于资源和优化选择等差异，需要执行不同的指令序列，但共享权重数据。cvimodel采用相应文件数据格式支持同一模型对多种batch和多种分辨率的推理。
- 模型分段：对于包含TPU不支持算子的模型，支持采用TPU和CPU协同方式进行推理计算。将一个模型分成若干段，每段由特定引擎（TPU或CPU）分别执行。
- 自定义算子：自定义算子目前仅支持采用CPU实现。在cvimodel中，用户自定义算子编译为特定平台的动态链接库并保存在cvimodel模型文件中。

> 为cv183x平台生成的cvimodel可以运行在1832/1835/1838等cv183x系统芯片上; 为cv182x平台生成的cvimodel可以运行在1821/1822/1826等cv182x系列芯片上；为cv181x平台生成的cvimodel可以运行在1810C/1811C/1812C/1810H/1811H/1812H等cv181x系列芯片上.

1.6 Runtime

> Runtime库和应用程序运行在CV183x/CV182x/CV181x SoC的ARM/aarch64处理器Linux系统中。Runtime提供一组API供应用程序运行时调用，实现模型在板端的在线推理。主要功能包括：

- 解析cvimodel文件；
- 加载权重数据、根据配置的batch_size和分辨率信息加载指令序列数据；
- 根据CPU段信息加载CPU函数；
- 加载输入数据；
- 执行推理计算；
- 返回结果数据。

1.6.1 Python Binding

> Runtime支持Python Binding，方便利用python的数据预处理和后处理代码快速进行模型开发和验证以及离线仿真。

1.6.2 仿真器

> Runtime除了调用硬件外，还支持以同样的API调用仿真器，进行离线测试和调试。

1.7 开发环境配置

推荐使用docker，具体使用方法请参考《TPU快速入门指南》中第三章"开发环境配置"。

第2章 模型编译指南

2.1 工具链参考

2.1.1 IR定义参考

2.1.1.1 概述

> IR（Intermediate Representation）即中间表示语言，其作用是将基于各类框架的神经网络图转换为统一的中间表示形式。Cvitek编译器借助了MLIR的IR框架，定义了面向TPU开发和优化的TPU Dialect。
>
> 每一个操作被定义一个Op，按照SSA规则定义，遵循如下原则：

- 每个Op有一个或者多个输入Tensor；
- 每个Op有且只有一个输出Tensor；
- 每个Tensor只有一个Op会改写它的值；
- 每个Op的操作除了会影响输出Tensor的值以外，不会产生任何其他副作用；

> 例如，对于SliceOp，虽然原本含义会有多个Output。但在TPU IR的定义中会对每一个sub tensor生成一个CropOp，它们的input tensor指向同一个tensor，通过attribute指定offset等参数，但每个CropOp只有一个输出tensor。

2.1.1.2 Operation支持列表

> 支持的基础Operation如下表所示：

操作	Engine(1)	Quantization(2)	Lowering(3)
BatchNorm	TPU	yes	yes
BroadcastMul	TPU	Yes	Yes
Clip	TPU	Yes	Yes
Concat	TPU	Yes	Yes
Conv2D	TPU	Yes	Yes
Crop	TPU	Yes	Yes
Custom	CPU	Yes	No
DeConv2D	TPU	Yes	Yes
DetectionOutput	CPU	No	No
EmbeddingOp	CPU	Yes	No
EltwiseAdd	TPU	Yes	Yes
EltwiseMax	TPU	Yes	Yes
EltwiseMul	TPU	Yes	Yes
FrcnDetection	CPU	No	No
FullyConnected	TPU	Yes	Yes
Gru	TPU	Yes	Yes
InstanceNorm	CPU	No	No
Interp	TPU	Yes	Yes
LayerNorm	TPU	Yes	Yes
LeakyRelu	TPU	Yes	Yes
Lstm	TPU	Yes	Yes
MatMul	TPU	Yes	Yes
Mish	TPU	Yes	Yes
Normalize	TPU	No	No
Pad	TPU	Yes	Yes
Permute	TPU	Yes	Yes
PixelShuffle	TPU	Yes	Yes
PoolAvg2D	TPU	Yes	Yes
PoolMask	TPU	No	No
PoolMax2D	TPU	Yes	Yes

操作	Engine(1)	Quantization(2)	Lowering(3)
Power	TPU	Yes	Yes
PRelu	TPU	Yes	Yes
PriorBox	CPU	No	No
Proposal	CPU	No	No
ReduceMax	TPU	Yes	Yes
ReduceMean	TPU	Yes	Yes
Relu	TPU	Yes	Yes
Reorg	TPU	Yes	Yes
RetinaFaceDetection	CPU	No	No
ROIPooling	CPU	Yes	No
Scale	TPU	No	No
ShuffleChannel	TPU	Yes	Yes
Sigmoid	TPU	Yes	Yes
Slice	TPU	Yes	Yes
Sqrt	TPU	Yes	Yes
Softmax	TPU	Yes	Yes
Std	TPU	Yes	Yes
TanH	TPU	Yes	Yes
Tile	TPU	Yes	Yes
Upsample	TPU	Yes	Yes
YoloDetection	CPU	No	No

> (1) Engine：来指定当前指令的执行阶段，TPU表示指令在TPU上执行，CPU表示在CPU上执行。

> (2) Quantization：表示是否需要做量化, 在CPU中执行的指令是不需要做量化的。

> (3) Lowering：表示当前指令需要转化为TPU指令，然后在TPU上执行。

2.1.1.3 CAFFE算子支持列表

算子	Engine	算子	Engine
BatchNorm	TPU	PReLU	TPU
BN	TPU	PriorBox	TPU
Concat	TPU	Proposal	CPU
Convolution	TPU	ReLU	TPU
ConvolutionDepthwise	TPU	ReLU6	TPU
Crop	TPU	Reorg	TPU
Deconvolution	TPU	Reshape	None
DetectionOutput	CPU	Reverse	TPU
Dropout	None	RetinaFaceDetection	CPU
DummyData	None	ROIPooling	CPU
Embed	CPU	Scale	TPU
Eltwise	TPU	ShuffleChannel	TPU
Flatten	None	Sigmoid	TPU
FrcnDetection	CPU	Slice	TPU
InnerProduct	TPU	Softmax	TPU
Input	None	Split	TPU
Interp	TPU	TanH	TPU
LRN	TPU	Tile	TPU
LSTM	TPU	Upsample	TPU
MatMul	TPU	YoloDetection	CPU
Mish	TPU		
Normalize	TPU		
Padding	TPU		
Permute	TPU		
Pooling	TPU		
Power	TPU		

> (1) None: 表示该OP会转换处理

2.1.1.4 ONNX算子支持列表

算子	Engine	算子	Engine
Abs	TPU	Min	TPU
Add	TPU	Mul	TPU
ArgMax	CPU	Neg	TPU
AveragePool	TPU	Pad	TPU
BatchNormalization	TPU	PRelu	TPU
Cast	TPU	Pow	TPU
Concat	TPU	Reciprocal	TPU
Conv	TPU	Relu	TPU
ConvTranspose	TPU	Reshape	None
Clip	TPU	Resize	TPU
Constant	TPU	ReduceL2	TPU
ConstantOfShape	TPU	ReduceMean	TPU
DepthToSpace	TPU	ReduceMax	TPU
Div	TPU	Shape	None
Dropout	None	Sigmoid	TPU
Equal	None	Slice	TPU
Elu	TPU	Softplus	TPU
Exp	TPU	Softmax	TPU
Expand	TPU	Split	TPU
Flatten	TPU	Squeeze	TPU
Gather	CPU	Sqrt	TPU
Gemm	TPU	Std	TPU
GlobalAveragePool	TPU	Sub	TPU
GlobalMaxPool	TPU	Sum	TPU
GRU	TPU	Tanh	TPU
HardSigmoid	TPU	Tile	TPU
Identity	None	Transpose	TPU
InstanceNormalization	CPU	Where	TPU
LeakyRelu	TPU	Unsqueeze	TPU
Log	TPU	YoloDetection	CPU

算子	Engine	算子	Engine
LRN	TPU		
LSTM	TPU		
LayerNorm	TPU		
MatMul	TPU		
MaxPool	TPU		
Max	TPU		

2.1.1.5 通用数据结构

- 基础数据类型

类型	描述
I8	DataType，8 bit整型数据
I32	DataType，32 bit整型数据
F32	DataType，32 bit浮点数据
BF16	DataType，16 bit BF浮点数据
I64	DataType，64 bit整型数据

- Tensor类型

类型	描述
Tensor	以DataType为数据类型的Tensor，不许为空
TensorOfOrNone	以DataType为数据类型的Tensor，None表示空Tensor
AnyTensor	以任意DataType为数据类型的Tensor
Variadic Tensor	一个或多个Tensor

- 基础属性类型

类型	描述
StrAttr	属性，字符串类型属性
NonNegativeI32Attr	属性，非负32 bit整数类型属性
I32Attr	属性，32 bit整数类型属性
F32Attr	属性，32 bit浮点类型属性
BoolAttr	属性，布尔属性

- TPU_QuantParamAttr

参数名称	类型	描述
mode	TPU_QuantModeAttr	Quant的类型
param_type	TPU_QuantParamTypeAttr	Quant变量类型
is_perchannel	BoolAttr	是为PerChannel，否为PerTensor
is_asymmetric	BoolAttr	是否为非对称两行
threshold_max	F32Attr	量化最大值
threshold_min	F32Attr	量化最小值（仅限非对称量化）
zero_point	I32Attr	零点值

- TPU_QuantModeAttr

枚举	描述
NONE	无量化，保持FP32
INT8	量化为INT8
BF16	量化为BF16

- TPU_QuantParamTypeAttr

枚举	描述
NONE	当前Op的量化无需变量
THRESHOLD	量化变量以Threshold方式描述
SCALE	量化变量以Scale描述，支持PerChannel或PerTensor
RSHIFT_ONLY	量化变量以RSHIFT描述，支持PerChannel或PerTensor
RSHIFT_AND_M_I32	量化变量以RSHIFT+I32 MULTIPLER描述，支持PerChannel
RSHIFT_AND_M_I8	量化变量以RSHIFT+I8 MULTIPLER描述，支持PerTensor
LUT_INT8	量化变量以INT8 LUT描述
LUT_BF16	量化变量以BF16 LUT描述

- TPU_ConvParamAttr

参数名称	类型	描述
stride_h	I32Attr	stride_h
stride_w	I32Attr	stride_w
padding	TPU_PaddingAttr	VALID或SAME
dilation_h	I32Attr	dilation_h
dilation_w	I32Attr	dilation_w
group	I32Attr	group
is_dw	BoolAttr	是否为Depthwise
with_bias	BoolAttr	是否有Bias
do_relu	BoolAttr	是否对结果进行relu操作
ins	I32ArrayAttr	对h， w插入0
pad_value	I32Attr	填充值

- TPU_PoolParamAttr

参数名称	类型	描述
kernel_h	I32Attr	kernel_h
kernel_w	I32Attr	kernel_w
padding_t	I32Attr	padding_t
padding_b	I32Attr	padding_b
padding_l	I32Attr	padding_l
padding_r	I32Attr	padding_r
stride_h	I32Attr	stride_h
stride_w	I32Attr	stride_w
do_relu	BoolAttr	是否对结果进行relu操作
count_include_pad	BoolAttr	计算时是否包含pad部分

2.1.1.6 Operation定义

- BatchNorm

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
mean	AnyTensor	mean参数向量	输入
variance	AnyTensor	variance参数向量	输入
scale	AnyTensor	scale参数	输入
variance_epsilon	F32Attr	epsilon	属性
name	StrAttr	名称	属性

- BroadcastMul

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
multiplier	AnyTensor	乘数向量	输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
do_relu	BoolAttr	是否对结果执行relu	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Clip

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	多个输入Tensor	多输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
min	F32Attr	最小值	属性
max	F32Attr	最大值	属性
name	StrAttr	名称	属性

- Concat

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	多个输入Tensor	多输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
axis	I32Attr	连接的axis	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Conv2D

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
filter	AnyTensor	Filter Tensor	输入
bias	TensorOfOrNone	Bias Tensor	输入(可选)
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
param	TPU_ConvParamAttr	Conv参数	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Crop

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
crop_offset	I32ArrayAttr	Crop Offset	属性
steps	I32ArrayAttr	步进	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Custom

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
operation_name	StrAttr	定制操作名字	属性
param	DictionaryAttr	操作所需参数	属性
tpu	BoolAttr	是否TPU处理	属性
do_quant	BoolAttr	是否需要量化	属性
threshold_overwrite	StrAttr	直接覆盖threshold	属性
name	StrAttr	名称	属性

- DeConv2D

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
filter	AnyTensor	Filter Tensor	输入
bias	TensorOfOrNone	Bias Tensor	输入(可选)
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
param	TPU_ConvParamAttr	Conv参数	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- DetectionOutput

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	输入Tensor	多输入
num_classes	I32Attr	检测类别数量	属性
share_location	BoolAttr	Share Location	属性
background_label_id	NonNegativeI32Attr	Background Label ID	属性
nms_threshold	F32Attr	NMS threshold	属性
top_k	I32Attr	Top K	属性
code_type	CodeTypeAttr	Code Type	属性
keep_top_k	I32Attr	Keep Top K	属性
confidence_threshold	F32Attr	Confidence Threshold	属性
name	StrAttr	名称	属性

- EltwiseAdd

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	多个输入Tensor	多输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
do_relu	BoolAttr	是否对结果执行relu	属性
do_early_stride	BoolAttr	是否提前执行stride	属性
early_stride_h	I32Attr	设置stride h	属性
early_stride_w	I32Attr	设置stride w	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- EltwiseMax

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	多个输入Tensor	多输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
do_relu	BoolAttr	是否对结果执行relu	属性
do_early_stride	BoolAttr	是否提前执行stride	属性
early_stride_h	I32Attr	设置stride h	属性
early_stride_w	I32Attr	设置stride w	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- EltwiseMul

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	多个输入Tensor	多输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
do_relu	BoolAttr	是否对结果执行relu	属性
do_early_stride	BoolAttr	是否提前执行stride	属性
early_stride_h	I32Attr	设置stride h	属性
early_stride_w	I32Attr	设置stride w	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- FrcnDetection

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	输入Tensor	多输入
class_num	I32Attr	检测类型数量	属性
obj_threshold	F32Attr	Object Threshold	属性
nms_threshold	F32Attr	NMS threshold	属性
keep_top_k	I32Attr	Keep Top K	属性
name	StrAttr	名称	属性

- FullyConnected

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
filter	AnyTensor	Filter Tensor	输入
bias	TensorOfOrNone	Bias Tensor	输入(可选)
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
do_relu	BoolAttr	是否对结果执行relu	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Gru

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
weight	AnyTensor	输入Weight	输入
recurrence	AnyTensor	输入Recurrence	输入
bias	TPU_TensorOfOrNone	输入Bias	输入
initial_h	AnyTensor	初始h	输入
sigmoid_table	TPU_TensorOfOrNone	sigmoid 表	输入
sigmoid_slope_table	TPU_TensorOfOrNone	sigmoid slop表	输入
tanh_table	TPU_TensorOfOrNone	tanh表	输入
tanh_slope_table	TPU_TensorOfOrNone	tanh slop 表	输入
quant	TPU_QuantParamAttr	Quant参数	属性
linear_before_reset	BoolAttr	在reset门之前有一个linear层	属性
bidirectional	BoolAttr	是否是bidirectional	
name	StrAttr	名称	属性

- LeakyRelu

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
quant_pos_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_pos_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_pos_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_pos_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
quant_neg_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_neg_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_neg_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_neg_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
negative_slope	F32Attr	负值斜率	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Lstm

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
weight	AnyTensor	输入Weight	输入
recurrence	AnyTensor	输入Recurrence	输入
bias	TPU_TensorOfOrNone	输入Bias	输入
initial_h	AnyTensor	初始h	输入
initial_c	AnyTensor	初始c	输入
sigmoid_table	TPU_TensorOfOrNone	sigmoid 表	输入
sigmoid_slope_table	TPU_TensorOfOrNone	sigmoid slop表	输入
tanh_table	TPU_TensorOfOrNone	tanh表	输入
tanh_slope_table	TPU_TensorOfOrNone	tanh slop 表	输入
quant	TPU_QuantParamAttr	Quant参数	属性
bidirectional	BoolAttr	是否是bidirectional	属性
name	StrAttr	名称	属性

- MatMul

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	输入Tensor	多输入
do_relu	BoolAttr	是否relu	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Mish

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	输入Tensor	多输入
table	TPU_TensorOfOrNone	lut表	输入
table_mantissa	TPU_TensorOfOrNone	mantissa表	输入
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Normalize

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
scale	AnyTensor	Scale Tensor	输入
across_spatial	BoolAttr	Across Spatial	属性
channel_shared	BoolAttr	Channel Shared	属性
name	StrAttr	名称	属性

- Pad

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	多输入
pads	I32ArrayAttr	填充的索引位置	属性
const_val	F32Attr	填充值	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Permute

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
order	I32ArrayAttr	Permute order	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- PixelShuffle

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
upscale_factor	NonNegativeI32Attr	Upscale factor	属性
quant	TPU_QuantParamAttr	Quant参数	属性
mode	DefaultValuedAttr	mode参数， 默认值是CRD	属性
name	StrAttr	名称	属性

- PoolAvg2D

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
quant_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
param	TPU_PoolParamAttr	Pool参数	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- PoolMask

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	多输入
pads	I32ArrayAttr	填充的索引位置	属性
const_val	F32Attr	填充值	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- PoolMax2D

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
param	TPU_PoolParamAttr	Pool参数	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Power

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
power	F32Attr	Power	属性
scale	F32Attr	Scale	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- PRelu

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
filter	AnyTensor	负值斜率向量	输入
quant_pos_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_pos_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_pos_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_pos_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
quant_neg_scale	TensorOfOrNone	量化scale向量	输入(可选)
quant_neg_zeropoint	TensorOfOrNone	量化zeropoint向量	输入(可选)
quant_neg_rshift	TensorOfOrNone	量化rshift向量	输入(可选)
quant_neg_multiplier	TensorOfOrNone	量化multiplier向量	输入(可选)
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- PriorBox

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
min_size	F32ArrayAttr	最小框大小	属性
max_size	F32ArrayAttr	最大框大小	属性
aspect_ratios	F32ArrayAttr	缩放宽高比	属性
variance	F32ArrayAttr	变量	属性
clip	BoolAttr	是否裁剪	属性
step_h	F32Attr	H维度的step	属性
step_w	F32Attr	W维度的step	属性
img_h	I32Attr	输入图像的高度	属性
img_w	I32Attr	输入图像的宽度	属性
offset	DefaultValuedAttr	默认框中心偏移量	属性
num_priors	I32Attr	默认框数目	属性
use_default_aspect_ratio	DefaultValuedAttr	是否使用默认宽高比	属性
name	StrAttr	名称	属性

- Proposal

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
net_input_h	I32Attr	网络的输入高度	属性
net_input_w	I32Attr	网络的输入宽度	属性
feat_stride	I32Attr	anchor box 的stride	属性
anchor_base_size	I32Attr	anchor 起始大小	属性
rpn_obj_threshold	F32Attr	候选框的可信度	属性
rpn_nms_threshold	F32Attr	NMS的可信度	属性
rpn_nms_post_top_n	I32Attr	保存NMS框数目	属性
name	StrAttr	名称	属性

- ReduceMax

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
quant	TPU_QuantParamAttr	Quant参数	属性
quant_scale	TPU_TensorOfOrNone	Quant收缩因子	属性
quant_zeropoint	TPU_TensorOfOrNone	Quant零点值	属性
quant_rshift	TPU_TensorOfOrNone	Quant右移位	属性
quant_multiplier	TPU_TensorOfOrNone	Quant乘数	属性
axes	OptionalAttr	指定维度	属性
name	StrAttr	名称	属性

- ReduceMean

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
quant	TPU_QuantParamAttr	Quant参数	属性
quant_scale	TPU_TensorOfOrNone	Quant收缩因子	属性
quant_zeropoint	TPU_TensorOfOrNone	Quant零点值	属性
quant_rshift	TPU_TensorOfOrNone	Quant右移位	属性
quant_multiplier	TPU_TensorOfOrNone	Quant乘数	属性
axes	OptionalAttr	指定维度	属性
name	StrAttr	名称	属性

- Relu

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
maximum	F32Attr	用于ReluM的最大值	属性(可选)
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Reorg

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
stride	NonNegativeI32Attr	宽和高的stride	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- RetinaFaceDetection

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	输入Tensor	多输入
nms_threshold	F32Attr	NMS threshold	属性
confidence_threshold	F32Attr	Confidence Threshold	属性
keep_top_k	I32Attr	Keep Top K	属性
name	StrAttr	名称	属性

- ROIPooling

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入

- Scale

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
scale	AnyTensor	scale向量	输入
bias	TensorOfOrNone	Bias向量	输入(可选)
do_relu	BoolAttr	是否对结果执行relu	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- ShuffleChannel

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
group	NonNegativeI32Attr	Shuffle Group	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Sigmoid

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
table	TensorOfOrNone	LUT table	输入(可选)
table_mantissa	TensorOfOrNone	LUT table mantissa	输入(可选)
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Slice

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
axis	I32Attr	切分的维度	属性
offset	I32Attr	在切分维度上的offset	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Sqrt

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
table	TensorOfOrNone	LUT table	输入(可选)
table_mantissa	TensorOfOrNone	LUT table mantissa	输入(可选)
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Softmax

参数名称 类型 描述 类别

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
exponential_table	TPU_TensorOfOrNone	exponential表	输入(可选)
reciprocal_table	TPU_TensorOfOrNone	Reciprocal表	输入(可选)
reciprocal_mantissa_table	TPU_TensorOfOrNone	reciprocal_mantissa表	输入(可选)
axis	I32Attr	Softmax的维度	属性
name	StrAttr	名称	属性

- TanH

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
table	TensorOfOrNone	LUT table	输入(可选)
table_mantissa	TensorOfOrNone	LUT table mantissa	输入(可选)
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- Tile

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
quant	TPU_QuantParamAttr	Quant参数	属性
quant_scale	TPU_TensorOfOrNone	Quant收缩因子	属性
quant_zeropoint	TPU_TensorOfOrNone	Quant零点值	属性
quant_rshift	TPU_TensorOfOrNone	Quant右移位	属性
quant_multiplier	TPU_TensorOfOrNone	Quant乘数	属性
resp	OptionalAttr	重复次数	属性
name	StrAttr	名称	属性

- Upsample

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	AnyTensor	输入Tensor	输入
mask	TPU_TensorOfOrNone	mask	输入(可选)
group	NonNegativeI32Attr	Shuffle Group	属性
quant	TPU_QuantParamAttr	Quant参数	属性
name	StrAttr	名称	属性

- YoloDetection

参数名称	类型	描述	类别
output	AnyTensor	输出Tensor	输出
input	Variadic Tensor	输入Tensor	多输入
net_input_h	I32Attr	网路输入分辨率h	属性
net_input_w	I32Attr	网路输入分辨率w	属性
class_num	I32Attr	检测类型数量	属性
nms_threshold	F32Attr	NMS threshold	属性
obj_threshold	F32Attr	Object Threshold	属性
keep_topk	I32Attr	Keep Top K	属性
spp_net	BoolAttr	Spp网络	属性
tiny_net	BoolAttr	Tiny网络	属性
yolo_v4_net	BoolAttr	Yolo_v4网络	属性
name	StrAttr	名称	属性

2.1.1.7 前端模型导入

> 工具链提供了python接口用于导入前端框架的IR到mlir模型中，所有的high level operation都定义在mlirimporter.py中，可以方便的构建mlir graph

【原型】

```
# mlirimporter.py
class MLIRImport:

  def __init__(self, inputs_shape, outputs_shape, input_type="FP32"):
    for input in inputs_shape:
      assert(isinstance(input, list))
      self.input_shape_list.append(input)
    for output in outputs_shape:
      assert(isinstance(output, list))
       self.output_shape_list.append(output)
    self.declare_func(input_type=input_type)
```

【主要属性】

```
MLIRImport.input_shape_list为模型的输入张量shape;

MLIRImport.output_shape_list为模型的输出张量shape。
```

【主要方法】

```
def add_input_op(self, name, index):
  pass
```

> 用于构造input指令，用来指定input的数据类型，threshold等属性。

功能说明	注释
返回值	Operation
name	指定input名字
index	指定input输入索引

```
def add_weight_fileOp(self, name):
  pass
```

> 用于构造weight操作，指定对应的weight文件。

功能说明	注释
返回值	Operation *
name	指定weight 文件名

```
def add_load_fileOp(self, name, output_tensor_shape,
                    tensor_type=TPU_TensorType.Fp32,
                    storage="NONE")
```

> 用于构造load_file操作,用来load weight相关的Tensor.

功能说明	注释
返回值	Operation *
name	Tensor名
output_tensor_shape	输出Tensor shape
tensor_type	Tensor类型
storage	存储类型

```
def add_conv_Op(self, op_name, inputOperands,
                output_tensor_shape,
                mode=TPU_MODE.FP32,
                **kargs)
```

> 用于构造convolution操作。

功能说明	注释
返回值	Operation *
op_name	指定conv层的名字
inputOperands	指定输入操作数
output_tensor_shape	指定输出shape
mode	指定数据类型
kargs	指定Conv的属性列表

> kargs字典序指定的参数如下

key	value
dilation_h	dilation_h值
dilation_w	dilation_w值
stride_h	stride_h值
stride_w	stride_w值
padding	VALID或SAME
padding_t	填充top值
padding_b	填充bottom值
padding_l	填充左侧值
padding_r	填充右侧值
group	group
is_dw	是否为Depthwise
with_bias	是否有bias
do_relu	是否对结果进行relu操作
ins	对h， w插入0

2.1.2 cvimodel文件格式参考

> cvimodel采用flatbuffers进行对权重数据、指令序列以及张量的相关信息进行打包封装，用于部署到平台上。

2.1.2.1 基本概念

- 模型（Model）：

> 为网络模型所有信息的集合，单个cvimodel中只能包含一个Model对象，但可以包含多个Batch的指令序列。

- 程序（Program）：

> 对应不同batch的指令序列。指令序列包含TPU段和CPU段，分别表示在TPU上运行的指令以及需要切换到CPU上运行的代码段。

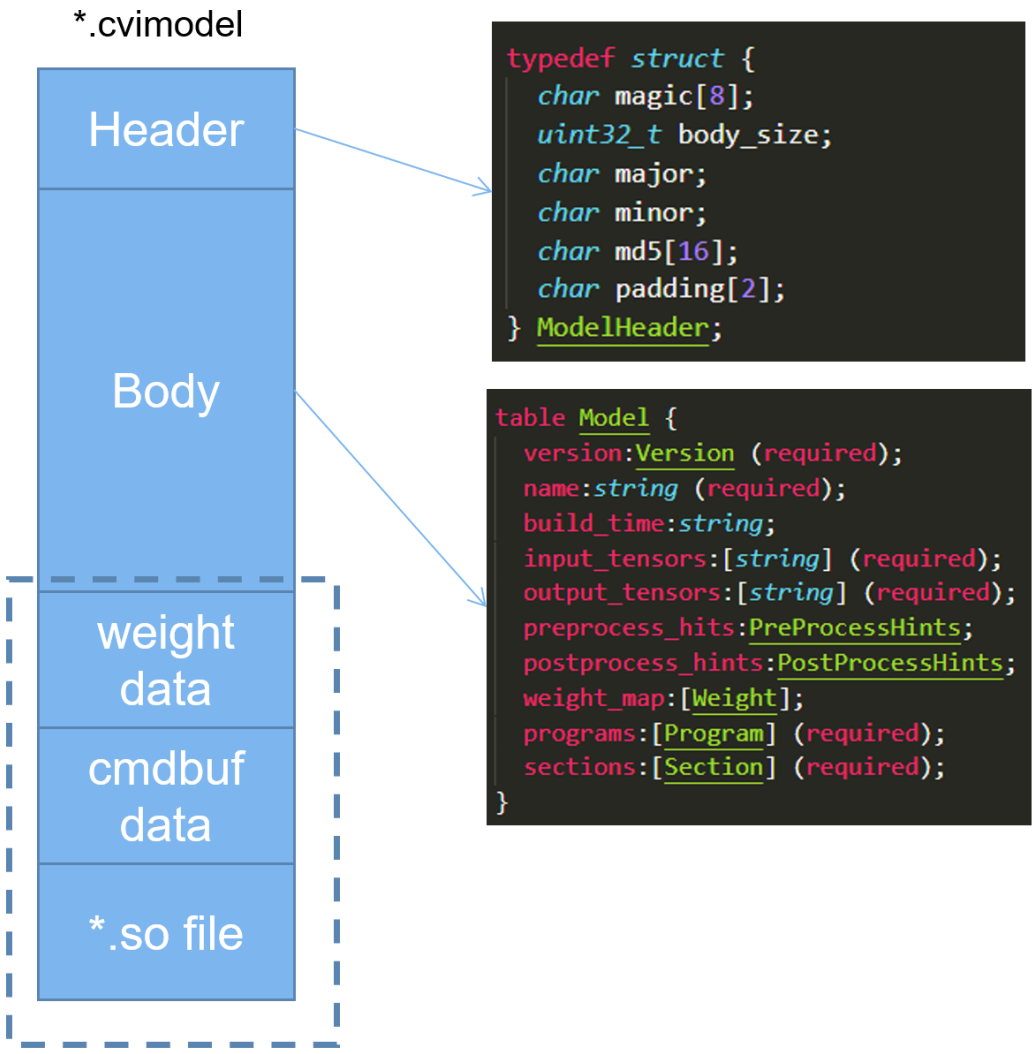
- 段（Routine）：

> 现分为TPU段和CPU段。单个Program中可能包含多个TPU段或者CPU段，在运行时会依序执行。

- 张量（Tensor）：

> 为输入输出张量和Activation等的统称，张量中包含其名称、Shape、基本数据类型等信息。

2.1.2.1 Cvimodel结构



> Cvimodel的基本结构如上图所示，分为三段。首段为cvimodel文件的header部分，包含magic字串，版本号，中段的数据字节数、md5值等数据，是解析cvimodel文件的基本信息；中段为Model的结构信息，包含Program、Routines等信息，用于解析网络模型的结构和指令信息；尾段为二进制数据段，包含权重数据，各Program的TPU指令序列，以及存储用户自定义CPU段的so文件。

2.1.2 工具链基本命令

2.1.2.1 model_transform

- 【命令】

```
model_transform.py [options] --mlir <output_mlir_file>
```

- 【作用】

> 将caffe/onnx模型转换为基于mlir的单精度模型，并做逐层精度比对验证，保证转模型的正确性。

- 【选项】

参数名称	描述
--model_type <type>	源模型的框架类型, 支持caffe, onnx等框架(pytorch,tensorflow需要先转为onnx)
--model_name <name>	模型的名字
--model_def <model_file>	模型文件(***.prototxt**, ***.onnx**等)
--model_data <caffemodel>	caffe模型权重文件(***.caffemodel**)
--image_resize_dims <h,w>	输入图片resize后的h和w, 如"256,256", 可选; 如果设置的image_resize_dims和net_input_dims不相等, 图片resize后还将center crop到net_input_dims指定的高宽; 如不设置, 则此值默认和net_input_dims相同
--keep_aspect_ratio <bool>	resize时是否保持原始高宽比不变，值为**1**或者**0**, 默认值为**0**; 如设置为**1**，在resize后高宽不足的部分会填充0
--net_input_dims <h,w>	模型的input shape的h与w: 如 **"224,224"**
--model_channel_order <order>	通道顺序，如**"bgr"** 或 **“rgb"**，默认值为**"bgr"**
--raw_scale <255.0>	raw_scale 默认值为**255**
--mean <0.0, 0.0, 0.0>	mean 通道均值，默认值为**"0.0 ,0.0 ,0.0"**, 值的顺序要和model_channel_order一致
--input_scale <1.0>	input_scale，默认值为**1.0**
--std <1.0 ,1.0 ,1.0>	std, 通道标准差，默认值为**"1.0 ,1.0 ,1.0"**, 值的顺序要和model_channel_order一致
--batch_size <num>	指定生成模型的的batch num，默认用模型本身的batch
--gray <bool>	是否输入的图片为灰度图，默认值为false
--image <image_file>	指定输入文件用于验证，可以是图片或npy；如果不指定，则不会做相似度验证
--tolerance <0.99,0.99,0.98>	mlir单精度模型与源模型逐层精度对比时所能接受的最小相似度, 相似度包括三项：余弦相似度、相关相似度、欧式距离相似度. 默认值为"0.99,0.99,0.98"
--excepts <"-">	逐层对比时跳过某些层, 多个层可以用逗号隔开, 如:"layer1,layer2", 默认值为"-",即对比所有层
--mlir <model_fp32_mlir>	输出mlir单精度模型

预处理过程用公式表达如下（x代表输入):

$$y = \frac{x \times \frac{raw\_scale}{255.0} - mean}{std} \times input\_scale$$

- 【示例】

```
model_transform.py \
    --model_type caffe \
    --model_name yolo_v3 \
    --model_def yolov3.prototxt \
    --model_data yolov3.caffemodel \
    --image_resize_dims '424,424' \
    --keep_aspect_ratio 1 \
    --net_input_dims '424,424' \
    --raw_scale 255 \
    --mean '0,0,0' \
    --input_scale 1.0 \
    --std '1,1,1'
    --model_channel_order 'rgb' \
    --batch_size 1 \
    --mlir yolo_v3_fp32.mlir
```

2.1.2.2 run_calibration

- 【命令】

```
run_calibration.py <model file> [option]
```

- 【作用】

> 针对mlir单精度模型，在测试集上随机挑选的images上做推理，统计各层的输出分布情况，计算出每层的threshold等值，输出一个threshold table用于后续的模型量化

- 【输入输出】

参数名称	描述
<model file>	输入mlir文件

- 【选项】

参数名称	描述
--dataset	指定校准图片集的路径
--image_list	指定样本列表，与dataset二选一
--input_num	指定校准图片数量
--histogram_bin_num	直方图bin数量, 默认为2048
--tune_num	指定微调使用的图片数量，增大此值可能会提升精度
--tune_thread_num	指定微调使用的线程数量，默认为4，增大此值可以减少运行时间，但是内存使用量也会增大
--forward_thread_num	指定模型推理使用的线程数量，默认为4，增大此值可以减少运行时间，但是内存使用量也会增大
--buffer_size	指定activations tensor缓存大小，默认为4G；如果缓存小于所有图片的activation tensor， 则增大此值会减少运行时间，若相反，则增大此值无效果
-o <calib_tabe>	输出calibration table文件

2.1.2.3 run_mix_precision

- 【命令】

> run_mix_precision.py <model_fp32_mlir> [options] -o <mix_precision_table>

- 【作用】

> 在测试集上计算每层量化为BF16后对于模型输出精度的收益，然后按照收益的大小排序，取靠前的n个层作为后面混精度量化中量化为BF16的层.

- 【选项】

  参数名称 描述

参数名称	描述
--dataset <path>	指定测试集的路径
--input_num <image_num>	指定测试图片的数量，默认为15张图
--image_list <image_list_file>	输入有测试图片路径列表的文件，用于指定固定数量的图片进行测试, 图片路径须为绝对路径
--calibration_table <calib_table>	输入calibration table
-o <mix_table>	输出mix precision table

2.1.2.4 model_deploy

- 【命令】

> model_deploy.py [options] <model_fp32_mlir> --cvimodel <out_cvimodel>

- 【作用】

> 进行int8或bf16量化、生成cvimodel文件；逐层验证量化模型与单精度模型的精度误差；验证cvimodel在仿真器上运行的结果是否正确.

- 【选项】

参数名称	描述
--model_name <name>	指定模型名
--mlir <model_fp32_mlir>	指定mlir单精度模型文件
--calibration_table <calib_table>	输入calibration table, 可选, 量化为int8模型
--mix_precision_table <mix_table>	输入mix precision table, 可选, 配合calib_table量化为混精度模型
--quantize <BF16>	指定默认量化方式，BF16/MIX_BF16/INT8
--tolerance <cos,cor,euc>	量化模型和单精度模型精度对比所能接受的最小相似度, 相似度包括三项：余弦相似度、相关相似度、欧式距离相似度.
--excepts <"-">	逐层对比时跳过某些层, 多个层可以用逗号隔开, 如:"layer1,layer2", 默认值为"-",即对比所有层
--correctness <cos,cor,euc>	cvimodel在仿真上运行的结果与量化模型推理的结果对比时所能接受的最小相似度, 默认值为:"0.99,0.99,0.98"
--chip <chip_name>	cvimodel被部署的目标平台名, 值为"cv183x"或"cv182x"或"cv181x"
--fuse_preprocess <bool>	是否在模型内部使用tpu做前处理（mean/scale/channel_swap等）
--pixel_format <format>	cvimodel所接受的图片输入格式, 详见下文
--aligned_input <bool>	cvimodel所接受的输入图片是否为vpss对齐格式, 默认值为false； 如果设置为true, 必须先设置fuse_process为true才能生效
--inputs_type <AUTO>	指定输入类型(AUTO/FP32/INT8/BF16/SAME)，如果是AUTO，当第一层是INT8时用INT8，BF16时用FP32
--outputs_type <AUTO>	指定输出类型(AUTO/FP32/INT8/BF16/SAME)，如果是AUTO，当最后层是INT8时用INT8，BF16时用FP32
--merge_weight <bool>	与同一个工作目前中生成的模型共享权重，用于后续合并同一个模型依据不同batch或分辨率生成的cvimodel
--model_version <latest>	支持选择模型的版本，默认为latest; 如果runtime比较老，比如1.2，则指定为1.2
--image <image_file>	用于测试相似度的输入文件，可以是图片、npz、npy；如果有多个输入，用,隔开
--save_input_files <bool>	保存model_deploy的指令及输入文件，文件名为${model_name}_deploy_files.tar.gz。 解压后得到的__deploy.sh保存了当前model_deploy的指令。

参数名称	描述
--dump_neuron <"-">	调试选项，指定哪些层可以在调用model_runner -i input.npz -m xxx.cvimodel --dump-all-tenors时， 可以dump下来，多个层可以用逗号隔开, 如:"layer1,layer2"，默认为-
--keep_output_name <bool>	保持输出节点名字与原始模型相同，默认为false
--cvimodel <out_cvimodel>	输出的cvimodel名

其中`pixel_format`用于指定外部输入的数据格式，有这几种格式:

pixel_format	说明
RGB_PLANAR	rgb顺序，按照nchw摆放
RGB_PACKED	rgb顺序，按照nhwc摆放
BGR_PLANAR	bgr顺序，按照nchw摆放
BGR_PACKED	bgr顺序，按照nhwc摆放
GRAYSCALE	仅有一个灰色通道，按nchw摆放
YUV420_PLANAR	yuv420 planner格式，来自vpss的输入
YUV_NV21	yuv420的NV21格式，来自vpss的输入
YUV_NV12	yuv420的NV12格式，来自vpss的输入
RGBA_PLANAR	rgba格式，按照nchw摆放

其中`aligned_input`用于表示是否数据存在对齐，如果数据来源于VPSS，则会有数据对齐要求，比如w按照32字节对齐。

2.1.2.5 model_runner

- 【命令】

> model_runner -i <inputs.npz> -m [options] -o <outputs.npz>

- 【作用】

> 通过仿真器运行生成的cvimodel，用于验证结果的正确性(比命令也可以在EVB上执行)

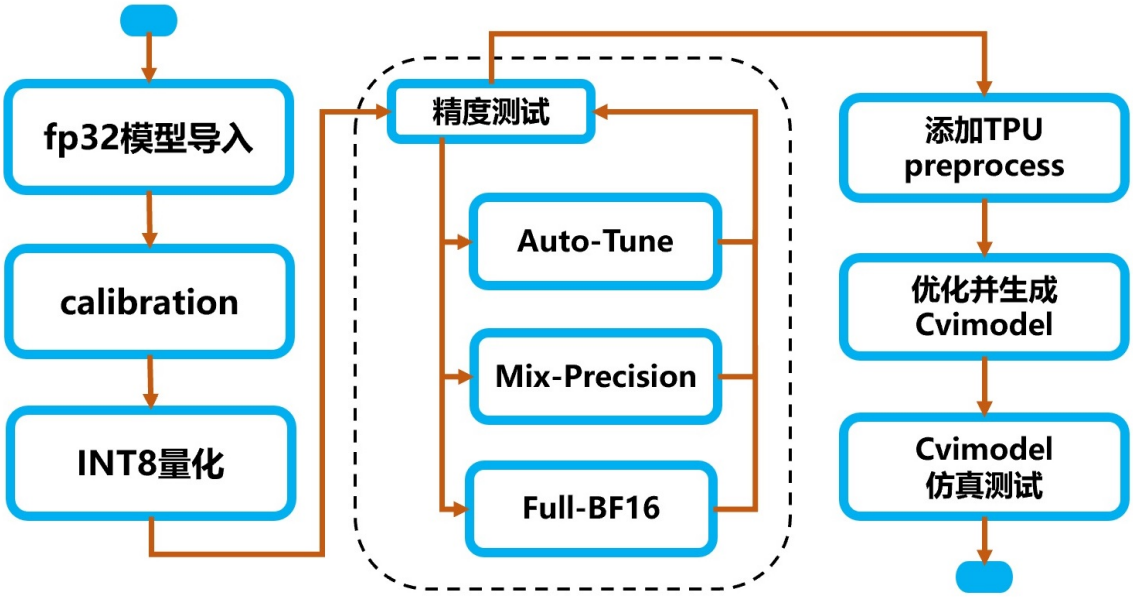
- 【选项】

参数名称　　描述

参数名称	描述
--model <cvimodel>	输入cvimodel模型文件, 必选项
--input <input_npz_file>	输入模型的参考输入npz文件，此文件由model_deploy.py生成, 如果不设置则使用随机值作为输入
--output <output_npz_file>	输出模型推理后的结果, 保存为npz文件, 可以透过numpy去读取, 可选项
--reference <reference_npz_file>	输入参考输出npz文件，此文件由model_deploy.py生成，可选项
--tolerance <cos,cor,euc>	输入模型输出和参考输出精度对比所能接受的最小相似度, 可选项. 相似度包括三项：余弦相似度、相关相似度、欧式距离相似度.
--program-id <int>	输入模型的program id，可选项
--count <int>	运行推理的次数，默认为1，可选项
--enable-timer	打印CVI_NN_Forward耗时

2.2 模型编译流程

> 一个模型从原训练框架保存的模型文件，编译为一个可以在TPURuntime执行推理的cvimodel需要经历如下图所述的过程。



具体流程定义如下：

2.2.1 FP32模型导入

> 模型导入阶段进行从训练框架模型文件到mlir描述模型的转换。工具链支持对Caffe，ONNX格式模型文件的导入。导入命令都由Python Interface实现。
>
> 模型导入过程主要分为三个部分：
>
> - 原模型推理，保存各层的输出到numpy npz文件
> - 原模型导入，将模型转换成FP32 MLIR模型
>   - FP32 MLIR模型推理，保存mlir模型各层的输出到numpy npz文件
>   - 输出对比，将两次推理各自产生的npz文件进行比对，确保模型转换正确
>   - FP32 MLIR模型图优化，优化后的mlir模型将作为后续流程的源文件

```
model_transform.py \
    --model_type $MODEL_TYPE \
    --model_name $NET \
    --model_def $MODEL_DEF \
    --model_data $MODEL_DAT \
    --image_resize_dims $IMAGE_RESIZE_DIMS \
    --net_input_dims $NET_INPUT_DIMS \
    --raw_scale $RAW_SCALE \
    --mean $MEAN \
    --input_scale $INPUT_SCALE \
    --model_channel_order $MODEL_CHANNEL_ORDER \
    --gray $GRAY \
    --batch_size $BATCH_SIZE \
    --mlir ${NET}_fp32.mlir
```

> 在此命令以及后续的命令中会对mlir模型的Activation输出进行三种相似度的比较：cosine similarity、correlation similarity、eulidean similarity以保证各阶段的精度损失在规定的范围内。

2.2.2 Calibration

Calibration的命令如下：

```
# 设置校正集目录
export DTATSET='xxxxxx/xxxx'
# 设置校正集随机选取的图片个数
export CALIBRATION_IMAGE_COUNT=1000

# 校正并生成calibration table
run_calibration.py \
    ${NET}_opt_fp32.mlir \
    --dataset=$DATASET \
    --input_num=${CALIBRATION_IMAGE_COUNT} \
    -o ${NET}_calibration_table
```

> Calibration是获取推理时各个tensor的统计信息过程。每个tensor的统计结果表现为一个threshold值，以及表征每个tensor动态范围的max和min两个值。
> Calibration工具采用python开发，需要输入量化数据集以及对应的模型前处理参数，结果会将每一层的统计threshold值导入到calibration table文件中。

2.2.3 INT8模型量化

> 模型的量化主要分为两种方式：
>
> - Per-Tensor量化（有时也称为Per-Layer）是指对整个Tensor使用同一组量化参数（scale或threshold);
> - Per-Channel量化（有时也称为Per-Axis）是指对于Channel这个维度支持各个Channel有各自不同的量化参数。
>
> 理论上，Weight Tensor和Activation Tensor都可以选择Per-Tensor或Per-Channel量化。但是实际实现过程中，CVITEK
> TPU选择只对Weight Tensor支持Per-Channel量化，对Activation Tensor保持Per-Tensor量化。
>
> 另外，按照INT8时0点映射的方法，量化方式也可以分为两种：
>
> - 对称量化，将需要映射的动态范围映射为正负对称的区间；
> - 非对称量化，是映射到非对称的区间，这时INT8的0点会被映射到一个非零的值；
>
> 在CVITEK工具链的量化工具中主要选择对称量化的方式，目前大部分模型使用此种方式都可取得良好的效果

模型INT8量化的命令如下：

```
# 加载calibration table,生成INT8 mlir模型
model_deploy.py \
    --model_name ${NET}_int8 \
    --mlir ${NET}_fp32.mlir \
    --calibration-table ${NET}_calibration_table \
    --mix_precision_table ${NET}_mix_table \
    --chip [cv183x|cv182x|cv181x] \
    --image ${IMAGE} \
    --tolerance ${INT8_QUANT_TOLERANCE} \
    --excepts ${EXCEPTS} \
    --cvimodel $CVIMODEL
```

> 在此命令mlir量化模型的Activation输出与mlir单精度模型进行三种相似度的比较：cosine similarity、correlation similarity、eulidean similarity以保证各阶段的精度损失在规定的范围内。

如果在逐层对比相似度通过后，特别是如果模型的输出层的相似度比较高时，可以进一步在测试集上验证mlir量化模型的精度。

> 工具链提供python binding用于mlir量化模型的推理，以及相应python工具支持对常见类型网络和常见数据集进行精度测试。用户可以基于python快速扩展，开发和对接自有数据集，后处理等流程。

如果在做相似度对比时的结果不理想，可以通过以下几种方式进行调整：

- 增加run_calibration.py时输入的图片数量。加大图片数量可以得到更加真实的各层activation的数据分布；
- 调整run_calibration.py的histogram_bin_num, 可以再扩大10倍或者更多。更大的histogram_bin_num可以使得统计各层activation直方图时的粒度更小，有利于得到更加真实的数据分布；
- 如果INT8模型的某些层的相似度掉很多时，可以手动调整calibration table文件中对应层的threshold值，可以加大或者减小，直到再次对比时取得比较好的相似度；

除此以外，还可以透过以下方式取得更好的精度：

2.2.4 Full-BF16量化

CVITEK TPU支持INT8和BF16两种数据格式运算的加速，在INT8精度不理想时，可以先尝试用将模型量化为BF16模型，再测试精度。其命令如下：

```
# SET_CHIP_NAME可依据部署的平台类型设置为cv183x或cv182x或cv181x
model_deploy.py \
    --model_name ${NET}_bf16 \
    --mlir ${NET}_fp32.mlir \
    --quantize bf16 \
    --chip [cv183x|cv182x|cv181x] \
    --image ${IMAGE_PATH} \
    --tolerance ${TOLERANCE_BF16} \
    --excepts ${EXCEPTS} \
    --correctness 0.99,0.99,0.99 \
    --cvimodel $cvimodel
```

2.2.5 Mix-Precision量化

全BF16模型的在CVITEK TPU上的性能会比全INT8模型的差。如果全BF16模型的精度能够满足要求，可以尝试混精度的方式对模型量化，以兼顾性能和精度。混精度的原理是在INT8模型的基础上，透过工具自动搜寻需要转换成BF16的层，然后将这些BF层输出到mix-precision table中。再导入mix-precision table后就可以生成混精度模型。具体命令如下：

```
# 可根据需要设置需要量化为BF16的层数
run_mix_precision.py \
    ${NET}_fp32.mlir \
    --dataset ${DATASET_PATH} \
    --input_num 15 \
    --max_bf16_layers $MIX_PRECISION_BF16_LAYER_NUM \
    --calibration_table ${NET}_calib_table.txt \
    -o ${NET}_mix_precision_table


model_deploy.py \
    --model_name ${NET}_bf16 \
    --mlir ${NET}_fp32.mlir \
    --calibration_table ${CALIBRATION_TABLE} \
    --mix_table ${MIX_PRECISION_TABLE} \
    --chip [cv183x|cv182x|cv181x] \
    --image ${IMAGE_PATH} \
    --tolerance ${TOLERANCE_BF16} \
    --excepts ${EXCEPTS} \
    --correctness 0.99,0.99,0.99 \
    --cvimodel $cvimodel
```

2.2.6 添加TPU Preprocessing

CVITEK TPU支持crop，减mean，乘scale以及channel swap等前处理操作。此步骤为可选项，可根据需要去添加TPU preprocessing；(由于CVITEK平台的图像处理硬件不支持输出BF16格式，所以对于BF16模型或者混精度模型中第一层为BF16层的模型需要添加TPU preprocessing)。具体命令如下：

```
# --pixel_format 设置输入数据的格式，目前支持:
#                RGB_PACKED, RGB_PLANAR,
#                BGR_PACKED, BGR_PLANAR,
#                GRAYSCALE, YUV420_PLANAR
model_deploy.py \
    --model_name ${NET}_int8 \
    --mlir ${NET}_fp32.mlir \
    --calibration_table ${NET}_tuned_calib_table \
    --mix_table ${NET}_mix_able \
    --chip [cv183x|cv182x|cv181x] \
    --image ${IMAGE_PATH} \
    --tolerance ${INT8_QUANT_TOLERANCE} \
    --excepts ${EXCEPTS} \
    --fuse_preprocess \
    --pixel_format BGR_PACKED \
    --cvimodel $cvimodel
```

> 生成cvimodel文件后，除可以在目标板runtime进行测试验证外，也可以调用仿真器进行离线测试验证。仿真器可以完全模拟硬件的推理精度。
>
> model_runner 是集成了runtime的binary工具，可以直接使用，也可以直接调用runtime API对cvimodel进行离线测试。离线测试的输出结果可以利用cvi_npz_tool.py compare进行数据比对。

第3章 Runtime开发指南

3.1 查看cvimodel

在runtime环境中部署cvimodel，请现在命令行中使用cvimodel_tool去查看cvimodel的详情，如输入、输出tensors的shape、name等，权重、Activations占用空间等信息，具体使用方法如下：

```
$ cvimodel_tool -a dump -i xxx.cvimodel
```

该命令的输出如下：

a. 版本信息部分：

```
Cvitek Runtime 1.2.0 # runtime lib的版本号
Mlir Version: tpu_rel_v1.3.4-42-g2bd9f2a54-dirty:20201205
# 编译此cvimodel所用工具链的版本号
Cvimodel Version: 1.2.0 # cvimodel的版本号
Build at 2020-12-05 23:37:09 # cvimodel编译的时间
CviModel Need ION Memory Size: (5.74 MB) # 模型会使用的ION内存大小
```

b. 权重和指令段

```
 Sections:

 ID   TYPE   NAME        SIZE   OFFSET ENCRYPT MD5
 000 weight weight       820800 0      False   49974c...
 001 cmdbuf tpu_func0_3e 1440   820800 False   caa513...
```

其中size为部署时weight或者cmdbuf(指令)所占用的memory的大小；encrypt表示该段是否为加密保存；MD5为该段数据的hash值，用于检查数据的完整性

c. 权重tensor列表

```
WeightMap:
ID  OFFSET SIZE   TYPE  N   C    H W NAME
000 1600   819200 int8  400 2048 1 1 filter_quant_lowered
001 0      1600   int32 400 1    1 1 bias_quant_lowered
```

d. program信息

> program对应于执行推理所需要的结构信息，具体包括Activations占用空间，模型的输入、输出tensor的名称，tpu子程序或者cpu子程序的详情以及tensor列表等信息

```
 Program %0
  batch_num : 1
  private_gmem_size: 0  # 私有memory大小
  shared_gmem_size: 448 # 共有memory\\<多个模型共享的内存区域\\>大小
  inputs: data_quant    # 模型输入tensor名称，可对应于后面的tensor列表
  outputs: input.7_Split_dequant # 模型输出tensor名称，可对应于后面的tensor列表

  routines: # program可有多个tpu或cpu子程序组成
    %000 tpu # tpu子程序
      inputs : data_quant
```

```
    outputs : input.7_Split_dequant
    section : tpu_func0_3e

tensor_map: # tensor列表
  ID  OFFSET TYPE N C H    W QSCALE   MEM    NAME
  000 0      int8 1 1 2048 1 5.314389 io_mem data_quant
  001 0      int8 1 1 400  1 -        shared fc
  002 10     int8 1 1 100  1 0.095460 shared input.7_Split
  003 0      fp32 1 1 100  1 -        io_mem input.7_Split_dequant
```

3.2 Runtime开发流程

模型加载

> Runtime对一个模型进行推理计算首先需要加载模型文件，runtime加载的对象为cvimodel文件。

获取输入输出Tensor

> 接下来，程序通过API分别获取Input Tensor和Output Tensor信息。对于支持多种batch_size的cvimodel，需要则会获取Tensor时指定batch_size。每个Tensor有自身的名称，类型，维度信息，以及存储空间。

执行推理

> 数据和buffer准备完毕后就可以开始推理计算。

预处理和后处理

> 预处理和后处理有几种处理方式。

- 应用程序自理：用户根据模型对前处理的需要，自行添加代码实现。
- 优化为预处理TPU段：在模型导入阶段，通过命令控制增加相应预处理或后处理操作。编译时，通过优化命令，对符合条件的预处理或后处理操作转换为TPU操作并编译进TPU Section中。运行时，随模型执行过程由TPU进行处理。

3.3 Runtime API参考

3.3.1 Runtime C API参考

头文件cviruntime.h中定义了runtime C API的数据结构和函数，用于模型的加载和推理，对应的动态库为libcviruntime.so，静态库为libcviruntime-static.a。

数据结构

> TPU Runtime涉及下列主要数据结构。

- CVI_FMT：张量数据的基本类型
- CVI_MEM_TYPE_E：张量数据的存储内存类型
- CVI_SHAPE：张量数据的维度
- CVI_MEM：张量数据的存储内存信息
- CVI_TENSOR：张量数据结构

- CVI_FRAME_TYPE：数据帧类型
- CVI_VIDEO_FRAME_INFO：视频帧数据结构
- CVI_MODEL_HANDLE：网络模型句柄
- CVI_CONFIG_OPTION：选项配置

CVI_FMT

```
typedef enum {
  CVI_FMT_FP32 = 0,
  CVI_FMT_INT32 = 1,
  CVI_FMT_UINT32 = 2,
  CVI_FMT_BF16 = 3,
  CVI_FMT_INT16 = 4,
  CVI_FMT_UINT16 = 5,
  CVI_FMT_INT8 = 6,
  CVI_FMT_UINT8 = 7
} CVI_FMT;
```

【描述】

TENSOR的基本数据类型

名称	描述
CVI_FMT_FP32	float 32类型
CVI_FMT_INT32	int32 类型
CVI_FMT_UINT32	uint32类型
CVI_FMT_BF16	bfloat16类型
CVI_FMT_INT16	int16类型
CVI_FMT_UINT16	uint16类型
CVI_FMT_INT8	int8类型
CVI_FMT_UINT8	uint8类型

CVI_MEM_TYPE_E

```
typedef enum {
  CVI_MEM_UNSPECIFIED = 0,
  CVI_MEM_SYSTEM      = 1,
  CVI_MEM_DEVICE      = 2
} CVI_MEM_TYPE_E;
```

【描述】

定义数据存储类型，表示数据存储的位置

名称	描述
CVI_MEM_UNSPECIFIED	初始值，表示未指定MEM内存来源
CVI_MEM_SYSTEM	MEM来自于系统内存
CVI_MEM_DEVICE	MEM来自于设备内存

CVI_SHAPE

```
#define CVI_DIM_MAX (6)

typedef struct {
  int32_t dim[CVI_DIM_MAX];
  size_t dim_size;
} CVI_SHAPE;
```

【描述】

定义TENSOR数据维度，按照n/channel/height/width四个维度排列.

名称　　描述

名称	描述
dim	各个维度大小
dim_size	维度数量，最多为6个维度

CVI_TENSOR

```
typedef struct {
  char            *name;
  CVI_SHAPE       shape;
  CVI_FMT         fmt;
  size_t          count;
  size_t          mem_size;
  uint8_t         *sys_mem;
  uint64_t        paddr;
  CVI_MEM_TYPE_E  mem_type;
  float           qscale;
  ...
} CVI_TENSOR;
```

【描述】

定义TENSOR结构体

名称	描述
name	tensor的名称
shape	tensor代表的维度
fmt	tensor的基本数据类型
count	tensor代表的元素个数
mem_size	tensor的所占用内存的大小
sys_mem	内存指针，指向系统内存
paddr	内存指针的物理地址
mem_type	tensor输入内存的类型
qscale	量化转换比例系数

CVI_FRAM_TYPE

```
typedef enum {
  CVI_NN_PIXEL_RGB_PACKED    = 0,
  CVI_NN_PIXEL_BGR_PACKED    = 1,
  CVI_NN_PIXEL_RGB_PLANAR     = 2,
  CVI_NN_PIXEL_BGR_PLANAR     = 3,
  CVI_NN_PIXEL_YUV_420_PLANAR = 13,
  CVI_NN_PIXEL_GRAYSCALE      = 15,
  CVI_NN_PIXEL_TENSOR         = 100,
  // please don't use below values,
  // only for backward compatibility
  CVI_NN_PIXEL_PLANAR         = 101,
  CVI_NN_PIXEL_PACKED        = 102
} CVI_NN_PIXEL_FORMAT_E;

typedef CVI_NN_PIXEL_FRAME_E CVI_FRAME_TYPE;
```

【描述】

定义输入数据的格式

名称	描述
CVI_NN_PIXEL_RGB_PACKED	RGB packed类型,格式为nhwc
CVI_NN_PIXEL_BGR_PACKED	BGR packed类型,格式为nhwc
CVI_NN_PIXEL_RGB_PLANAR	RGB planar类型,格式为nchw
CVI_NN_PIXEL_BGR_PLANAR	BGR planar类型,格式为nchw
CVI_NN_PIXEL_YUV_420_PLANAR	YUV420 planar类型
CVI_NN_PIXEL_GRAYSCALE	灰度图, YUV400
CVI_NN_PIXEL_TENSOR	紧密排布的4维度张量(1.3版本前默认类型

CVI_VIDEO_FRAME_INFO

```
typedef struct {
  CVI_FRAME_TYPE type;
  CVI_SHAPE shape;
  CVI_FMT fmt;
  uint32_t stride[3];
  uint64_t pyaddr[3];
} CVI_VIDEO_FRAME_INFO;
```

【描述】

定义数据帧类型

名称	描述
type	数据帧类型
shape	数据帧的维度数据
fmt	基本数据类型
stride	frame w维度的间隔，对齐到字节
pyaddr	通道的物理地址，当type为PLANAR时需要填入每个通道的地址；当type为PACKED时，只用填数据首地址

CVI_MODEL_HANDLE

```
typedef void *CVI_MODEL_HANDLE;
```

【描述】

神经网络模型句柄，通过接口CVI_NN_RegisterModel得到该句柄，并由接口

CVI_NN_CleanupModel释放句柄的资源。

CVI_CONFIG_OPTION

```
typedef enum {
  OPTION_OUTPUT_ALL_TENSORS = 4,
  OPTION_PROGRAM_INDEX      = 9
} CVI_CONFIG_OPTION;
```

【描述】

定义CVI_NN_SetConfig接口获取或者设置模型配置的枚举类型:

名称	类型	默认值	描述
OPTION_PRGRAM_INDEX	int	0	配置推理模型的program index, cvimodel可以存放模型多个batch size或者多种分辨率的指令(或称为program), 通过program id可以选择执行对应的program
OPTION_OUTPUT_ALL_TENOSRS	bool	false	配置runtime将模型所有可见的TENSOR作为模型的输出，则选项可以作为debug的手段之一

返回码

```
#define CVI_RC_SUCCESS 0 // The operation was successful
#define CVI_RC_AGAIN 1   // Not ready yet
#define CVI_RC_FAILURE 2 // General failure
#define CVI_RC_TIMEOUT 3 // Timeout
#define CVI_RC_UNINIT 4  // Uninitialzed
#define CVI_RC_INVALID_ARG 5 // Arguments invalid
#define CVI_RC_NOMEM 6       // Not enough memory
#define CVI_RC_DATA_ERR 7    // Data error
#define CVI_RC_BUSY 8        // Busy
#define CVI_RC_UNSUPPORT 9   // Not supported yet
typedef int CVI_RC;
```

【描述】

返回码用于表示接口执行结果是否存在异常，其中CVI_RC_SUCCESS为成功，其他值为失败。

函数

> TPU Runtime提供下述基本接口。

- CVI_NN_RegisterModel：从文件中加载神经网络模型
- CVI_NN_RegisterModelFromBuffer：从内存中加载网络模型
- CVI_NN_RegisterModelFromFd：从fd加载网络模型
- CVI_NN_CloneModel: 复制神经网络模型
- CVI_NN_SetConfig: 配置神经网络模型
- CVI_NN_GetInputOutputTensors：获取输入以及输出TENSOR的信息
- CVI_NN_Forward：神经网络推理，同步接口
- CVI_NN_ForwardAsync：神经网络推理，异步接口
- CVI_NN_ForwardWait：等待推理任务完成
- CVI_NN_CleanupModel：释放网络资源
- CVI_NN_GetTensorByName：根据名字获取张量信息
- CVI_NN_TensorPtr：获取张量的系统内存指针
- CVI_NN_TensorSize：获取张量的系统内存大小
- CVI_NN_TensorCount：获得张量的元素个数
- CVI_NN_TensorQuantScale：获得张量的量化系数
- CVI_NN_TensorShape：获得张量的Shape
- CVI_NN_SetTensorPtr：设置张量的系统内存
- CVI_NN_SetTensoPhysicalAddrr：设置张量的物理内存
- CVI_NN_SetTensorWithAlignedFrames：将视频帧数据拷贝到张量

- CVI_NN_SetTensorWithVideoFrames: 将视频帧数据拷贝到张量

CVI_NN_RegisterModel

【原型】

```
CVI_RC CVI_NN_RegisterModel(
    const char *model_file,
    CVI_MODEL_HANDLE *model)
```

【描述】

> 从文件中加载cvimodel，并返回模型句柄。此句柄将作为后续API调用的接口的参数。当模型不再使用时，需要调用CVI_NN_CleanupModel接口对模型资源进行释放。

参数名称	描述	输入/输出
Model_file	cvimodel模型文件名	输入
model	网络模型句柄	输出

CVI_NN_RegisterModelFromBuffer

【原型】

```
CVI_RC CVI_NN_RegisterModelFromBuffer(
    const int8_t *buf,
    uint32_t size,
    CVI_MODEL_HANDLE *model);
```

【描述】

> 从内存中加载cvimodel，并返回模型句柄。此句柄将作为后续API调用的接口的参数。当模型不再使用时，需要调用CVI_NN_CleanupModel接口对模型资源进行释放。

参数名称	描述	输入/输出
buf	内存地址	输入
size	模型的内存大小	输入
model	网络模型句柄	输出

CVI_NN_RegisterModelFromFd

【原型】

```
CVI_RC CVI_NN_RegisterModelFromFd(
    const int32_t fd,
    const size_t ud_offset,
    CVI_MODEL_HANDLE *model);
```

【描述】

> 从fd的offset位置加载cvimodel，并返回模型句柄。此句柄将作为后续API调用的接口的参数。当模型不再使用时，需要调用CVI_NN_CleanupModel接口对模型资源进行释放。

参数名称	描述	输入/输出
fd	文件描述符	输入
offset	加载模型的位置	输入
model	网络模型句柄	输出

CVI_NN_CloneModel

【原型】

```
CVI_RC CVI_NN_CloneModel(
    CVI_MODEL_HANDLE model,
    CVI_MODEL_HANDLE *cloned)
```

【描述】

> 当需要运行同一个cvimodel模型的不同或者不同batch size指令时，可以调用此接口复制模型，复制后的模型句柄将和之前句柄共享部分资源，可以有效的减少系统内存开销。该句柄在不使用后，也需通过CVI_NN_CleanupModel进行释放。

参数名称 描述 输入/输出

参数名称	描述	输入/输出
model	已经存在模型句柄	输入
cloned	返回复制的模型句柄	输出

CVI_NN_SetConfig

【原型】

```
CVI_RC CVI_NN_SetConfig(
    CVI_MODEL_HANDLE model,
    CVI_CONFIG_OPTION option,
    ...)
```

【描述】

> 用于对模型模型进行配置，可供配置的选项请参考CVI_CONFIG_OPTION。如果不需要改变配置的默认值，则可不必调用。

注意，此接口需要在CVI_NN_GetInputOutputTensors之前调用。

参数名称 描述 输入/输出

参数名称	描述	输入/输出
model	模型句柄	输入
option	配置选项	输入
可变参数	根据option的类型传入配置值	输入

【示例】

```
CVI_NN_SetConfig(model, OPTION_BATCH_SIZE, 1);
CVI_NN_SetConfig(model, OPTION_OUTPUT_ALL_TENSORS, false);
```

CVI_NN_GetInputOutputTensors

【原型】

```
CVI_RC CVI_NN_GetInputOutputTensors(
    CVI_MODEL_HANDLE model,
    CVI_TENSOR **inputs, int32_t *input_num,
    CVI_TENSOR **outputs, int32_t *output_num)
```

【描述】

> 获取输入和输出TENSOR的信息，同时给TENSOR分配内存。

参数名称	描述	输入/输出
model	模型句柄	输入
inputs	输入TENSOR数组	输出
input_num	输入TENSOR个数	输出
outputs	输出TENSOR数组	输出
output_num	输出TENSOR个数	输出

CVI_NN_Forward

【原型】

```
CVI_RC CVI_NN_Forward(
     CVI_MODEL_HANDLE model,
     CVI_TENSOR inputs[], int32_t input_num,
     CVI_TENSOR outputs[], int32_t output_num);
```

【描述】

> 模型前向推理。此接口为阻塞型，会阻塞直至前向推理完成。inputs和outputs必须已经申请过buffer，且输入数据已经存储在inputs的buffer中，推理的结果会保存在outputs的buffer中。

参数名称	描述	输入/输出
model	网络模型句柄	输入
inputs	输入tensors	输入
input_num	输入tensor的数量	输入
outputs	输出tensors	输出
output_num	输出tensors的数量	输入

CVI_NN_CleanupModel

【原型】

```
CVI_RC CVI_NN_CleanupModel(CVI_MODEL_HANDLE model)
```

【描述】

> 释放模型所有资源。如果模型被复制过，则此接口只会将模型的引用次数递减。当引用次数为0时，才会释放模型所有资源。

参数名称	描述	输入/输出
model	网络模型句柄	输入

CVI_NN_GetTensorByName

【原型】

```
CVI_TENSOR *CVI_NN_GetTensorByName(
    const char *name,
    CVI_TENSOR *tensors,
    int32_t num)
```

【描述】

根据名称从tensors中获得对应的tensor指针。

参数名称	描述	输入/输出
name	Tensor的名称;可以指定为 CVI_NN_DEFAULT_TENSOR, input_num为1时返回唯一的那个tensor，否则返回NULL	输入
tensors	tensors数组	输入
num	tensor的个数	输入

CVI_NN_TensorPtr

【原型】

```
void *CVI_NN_TensorPtr(CVI_TENSOR *tensor)
```

【描述】

> 获得TENSOR中的系统内存指针。

参数名称	描述	输入/输出
tensor	tensor指针	输入

CVI_NN_TensorSize

【原型】

```
size_t CVI_NN_TensorSize(CVI_TENSOR *tensor);
```

【描述】

> 获得tensor的字节大小。

参数名称	描述	输入/输出
tensor	tensor指针	输入

CVI_NN_TensorCount

【原型】

```
size_t CVI_NN_TensorCount(CVI_TENSOR *tensor);
```

【描述】

> 获得TENSOR的元素个数。

参数名称	描述	输入/输出
tensor	tensor指针	输入

CVI_NN_TensorQuantScale

【原型】

```
float CVI_NN_TensorQuantScale(CVI_TENSOR *tensor)
```

【描述】

> 获得TENSOR的量化系数，用于fp32到int8的转化

参数名称	描述	输入/输出
tensor	tensor指针	输入

CVI_NN_TensorShape

【原型】

```
CVI_SHAPE CVI_NN_TensorShape(CVI_TENSOR *tensor)
```

【描述】

> 获得TENSOR的Shape。

参数名称	描述	输入/输出
tensor	tensor指针	输入

CVI_NN_SetTensorPtr

【原型】

```
CVI_RC CVI_NN_SetTensorPtr(
    CVI_TENSOR *tensor,
    uint8_t *buf)
```

【描述】

> 设置TENSOR的buffer内存。

参数名称	描述	输入/输出
tensor	tensor指针	输入
buf	系统内存指针	输入

CVI_NN_SetTensorPhysicalAddr

【原型】

```
CVI_RC CVI_NN_SetTensorPhysicalAddr(
    CVI_TENSOR *tensor,
    uint64_t *paddr)
```

【描述】

> 设置TENSOR的物理地址。调用该接口后，会释放TENSOR初始化时自动分配的内存。

参数名称	描述	输入/输出
tensor	tensor指针	输入
paddr	ion内存指针	输入

CVI_NN_SetTensorWithAlignedFrames

【原型】

```
CVI_RC CVI_NN_SetTensorWithAlignedFrames(
    CVI_TENSOR *tensor,
    uint64_t frame_paddrs[],
    int32_t frame_num,
    CVI_NN_PIXEL_FORMAT_E pixel_format);
```

【描述】

> 将一帧或者多帧数据拷贝到张量。

参数名称	描述	输入/输出
tensor	tensor指针	输入
frame_paddrs	视频帧的物理地址数组	输入
frame_num	视频帧的个数	输入
pixel_format	视频帧格式类型	输入

CVI_NN_SetTensorWithVideoFrame

【原型】

```
CVI_RC CVI_NN_SetTensorWithVideoFrame(
    CVI_MODEL_HANDLE model, CVI_TENSOR* tensor,
    CVI_VIDEO_FRAME_INFO* video_frame_info);
```

【描述】

> 将视频帧数据拷贝到张量

参数名称	描述	输入/输出
model	模型句柄	输入
tensor	tensor指针	输入
video_frame_info	视频帧信息	输入

3.3.2 Runtime Python API

> Runtime通过pybind11将底层Runtime C++代码封装为Python API。Runtime Python API目前支持在python3.6环境下执行。其主要API如下所示：

pyruntime.Tensor

Tensor表示张量对象，

【原型】

```
class Tensor:

  def __init__(self):
    self.name = str()
    self.data = numpy.ndarray()
```

【属性】

> Tensor.name为张量的名称；
>
> Tensor.data为numpy arrary用于存放张量的数据。可以分别用data.shape和data.dtype获取张量的Shape以及基本数据类型。

pyruntime.Model

> Model表示模型对象

【原型】

```
class Model:

  def __init__(self, model_file, batch_num=0, dump_all_tensors=False):
    self.inputs = [Tensor]
    self.outputs = [Tensor]

  def forward(self):
    pass
```

【属性】

> Model.inputs 为模型的输入张量(pyruntime.Tensor)数组；
>
> Model.outputs为模型的输出张量(pyruntime.Tensor)数组。

【方法】

```
def __init__(self, model_file, batch_num=0, dump_all_tensors=False)
```

> Model类的构造方法，用于注册并配置cvimodel模型文件。

功能说明	注释
返回值	None
batch_size	Int型, 指定模型的batch size
dump_all_tensors	Bool型, 将网络可见的张量作为输出Tensors

```
def forward(self)
```

> 用于做模型的前向推理

功能说明	注释
返回值	None

示例

```
 import pyruntime

 # initialize the cvimodel
 model = pyruntime.Model("1.cvimodel")
 if model == None:
   raise Exception("cannot load cvimodel")

# fill data to inputs
 data = model.inputs[0].data
 input_data = np.fromfile("input.bin", dtype=data.dtype)
                .reshape(data.shape)
 data[:] = input_data

 # forward
 model.forward()

 # get output date
 for out in model.outputs:
   print(outputs)
```

3.4 Runtime日志设置

Runtime日志路径

> /var/log/tpu

Runtime日志配置

> 以修改输出日志级别为info为例:
>
> 修改/etc/rsyslog.d/tpu.conf文件
>
> 输入:

> if $syslogfacility-text == "local6" and ($syslogseverity <= 6) then /dev/console （日志级别小于等于6则输出到终端）
>
> if $syslogfacility-text == "local6" and ($syslogseverity <= 6) then /var/log/tpu（日志级别小于等于6则输出到/var/log/tpu文件中）
>
> 然后使用命令
>
> /etc/init.d/P01rsyslog restart
>
> 即可生效

在开发板上输入

syslogd -l 8 -O /dev/console

或

busybox syslogd -l 8 -O /dev/console

会输出所有runtime日志

Runtime日志级别

FATAL	**0**
ERROR	3
WARNING	4
NOTICE	5
INFO	6
DEBUG	7