CVITEK TPU快速入门指南

> 文档版本: 1.5.10
>
> 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。

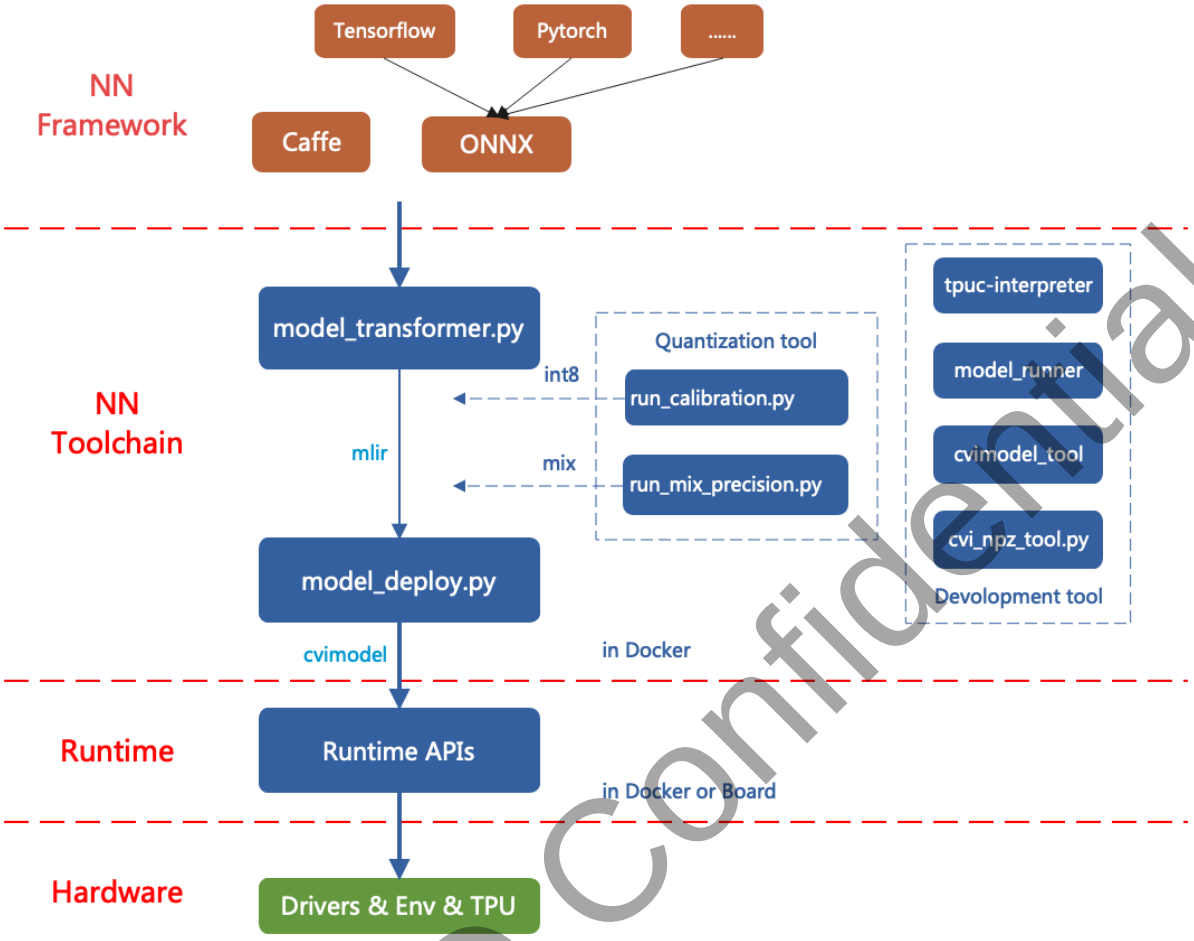
本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。

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

1 概述

1) 整理框架介绍



2) Release 内容

CVITEK Release包含如下组成部分：

文件	描述
cvitek_mlir_ubuntu-18.04.tar.gz	cvitek NN工具链软件
cvitek_tpu_sdk.tar.gz	cvitek Runtime SDK，包括交叉编译头文件和库文件
cvitek_tpu_samples.tar.gz	sample程序源代码
cvimodel_samples.tar.gz	sample程序使用的cvimodel模型文件
docker_cvitek_dev_1.8-ubuntu-18.04.tar	cvitek docker镜像文件

3) 经过验证的网络列表

- **Classification:** `resnet50` `resnet18` `mobilenet_v1` `mobilenet_v2` `squeezenet_v1.1` `shufflenet_v2` `googlenet` `inception_v3` `inception_v4` `vgg16` `densenet_121` `densenet_201` `senet_res50` `resnext50` `res2net50` `ecanet50` `efficientnet_b0` `efficientnet_lite_b0` `nasnet_mobile`
- **Detection:** `retinaface_mnet25` `retinaface_res50` `ssd300` `mobilenet_ssd` `yolo_v1` `yolo_v2` `yolo_v3` `yolo_v4` `yolo_v5` `yolo_x`
- **Misc:** `arcface_res50` `alphapose` `espcn_3x` `unet` `erfnet`

2 开发环境配置

加载镜像文件：

```
docker load -i docker_cvitek_dev_1.8-ubuntu-18.04.tar
```

或者从docker hub获取:

```
docker pull cvitek/cvitek_dev:1.8-ubuntu-18.04
```

如果是首次使用docker，可执行下述命令进行安装和配置（Ubuntu系统）

```
sudo apt install docker.io
systemctl start docker
systemctl enable docker

sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker (use before reboot)
```

取得docker image后，执行下述命令运行docker：

```
docker run -itd -v $PWD:/work --name cvitek cvitek/cvitek_dev:1.8-ubuntu-18.04
docker exec -it cvitek bash
```

如果需要挂载模型和数据集到docker中，可以如下操作：（可选）

```
# 这里假设models和dataset分别位于~/data/models和~/data/dataset目录，如有不同请相应调整。
docker run -itd -v $PWD:/work \
   -v ~/data/models:/work/models \
   -v ~/data/dataset:/work/dataset \
   --name cvitek cvitek/cvitek_dev:1.8-ubuntu-18.04
docker exec -it cvitek bash
```

3 编译移植pytorch模型

本章以resnet18为例，介绍如何编译迁移一个pytorch模型至CV183x TPU平台运行。

本章需要如下文件：

- cvitek_mlir_ubuntu-18.04.tar.gz

除caffe外的框架，如tensorflow/pytorch均可以参考本章节步骤，先转换成onnx，再转换成cvimodel。

如何将模型转换成onnx，可以参考onnx官网: https://github.com/onnx/tutorials

步骤 0：加载cvitek_mlir环境

```
tar zxf cvitek_mlir_ubuntu-18.04.tar.gz
source cvitek_mlir/cvitek_envs.sh
```

步骤 1：获取pytorch模型并转换为onnx

使用torchvision提供的resnet18模型https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

使用下列python脚本下载pytorch模型，并将pytorch模型输出为onnx格式，保存在`model_resnet18`目录：

```
mkdir model_resnet18
cd model_resnet18
```

执行python命令：

```
# python
import torch
import  torchvision.models as models
# Use an existing model  from Torchvision, note it
# will download this if  not already on your computer (might take time)
model = models.resnet18(pretrained=True)
# Create some sample  input in the shape this model expects
dummy_input = torch.randn(1, 3, 224, 224)
# Use the exporter from  torch to convert to onnx
torch.onnx.export(model,
    dummy_input,
    'resnet18.onnx',
    export_params=True,
    opset_version=13,
    verbose=True,
    input_names=['input'])
```

得到`resnet18.onnx`。

步骤 2：onnx模型转换为fp32 mlir形式

创建工作目录workspace，拷贝测试图片cat.jpg，和数据集100张图片（来自ILSVRC2012）：

```
mkdir workspace && cd workspace
cp $MLIR_PATH/tpuc/regression/data/cat.jpg .
cp -rf $MLIR_PATH/tpuc/regression/data/images .
```

推理前，我们需要了解这个模型的预处理参数，resnet18的预处理如链接描述https://pytorch.org/hub/pytorch_vision_resnet：

> preprocess = transforms.Compose([
> transforms.Resize(256),
> transforms.CenterCrop(224),
> transforms.ToTensor(),
> transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
>])

使用`model_transform.py`将onnx模型转换成mlir文件，其中可支持预处理参数如下:

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npz、npy（w-major order）； 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）：

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

由onnx模型转换为mlir,执行以下shell：

```
model_transform.py \
  --model_type onnx \
  --model_name resnet18 \
  --model_def ../resnet18.onnx \
  --image ./cat.jpg \
  --image_resize_dims 256,256 \
  --keep_aspect_ratio false \
  --net_input_dims 224,224 \
  --raw_scale 1.0 \
  --mean 0.485,0.456,0.406 \
  --std 0.229,0.224,0.225 \
  --input_scale 1.0 \
  --model_channel_order "rgb" \
  --tolerance 0.99,0.99,0.99 \
  --mlir resnet18_fp32.mlir
```

得到resnet18_fp32.mlir文件.

注： 上述填入的预处理参数仅仅以信息的形式存放在mlir中，后续转换成cvimodel，也仅以信息的方式存放。对图片的预处理过程需要再外部处理，再传给模型运算。如果需要模型内部对图片进行预处理，请参考第8章节：使用TPU做前处理。

步骤 3：生成全bf16量化cvimodel

```
model_deploy.py \
  --model_name resnet18 \
  --mlir resnet18_fp32.mlir \
  --quantize BF16 \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.99,0.99,0.86 \
  --correctness 0.99,0.99,0.93 \
  --cvimodel resnet18_bf16.cvimodel
```

步骤 4：生成全int8量化cvimodel

先做Calibration，需要先准备校正图片集,图片的数量根据情况准备100~1000张左右。

这里用100张图片举例，执行calibration,执行如下shell：

```
run_calibration.py \
  resnet18_fp32.mlir \
  --dataset=./images \
  --input_num=100 \
  -o resnet18_calibration_table
```

得到`resnet18_calibration_table`。这里用`--dataset`指定样本目录，也可以用`--image_list`指定样本列表。

run_calibration.py 相关参数说明如下：

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

导入calibration_table，生成cvimodel：

```
model_deploy.py \
  --model_name resnet18 \
  --mlir resnet18_fp32.mlir \
  --calibration_table resnet18_calibration_table \
  --quantize INT8 \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.98,0.98,0.84 \
  --correctness 0.99,0.99,0.99 \
  --cvimodel resnet18_int8.cvimodel
```

model_deploy.py的相关参数说明如下：

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（w-major order）；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。

步骤 5：开发板中测试模型性能数据

配置开发板的TPU sdk环境：

```
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

测试仿真环境与真实硬件的输出结果，需要步骤三或步骤四生成的调试文件：

- xxx_quantized_tensors_sim.npz 仿真环境中网络推理过程的tensor文件，作为与真实硬件输出结果参考对比
- xxx_in_fp32.npz 模型的输入tensor文件，测试不同类型的模型，input_npz文件需要不一样
- xxx_[int8/bf16].cvimodel 输出int8或者bf16的cvimodel文件

在开发板中执行以下shell，测试量化为int8模型的仿真环境和真实硬件输出结果进行比较：

```
model_runner \
--input resnet18_in_fp32.npz \
--model resnet18_int8.cvimodel \
--output out.npz \
--reference resnet18_quantized_tensors_sim.npz
```

若需打印性能数据, 可在执行`model_runner`之前定义如下环境变量（仅在开发板中生效）：

```
export TPU_ENABLE_PMU=1
```

运行时会出现如下日志：

```
=======================inference total info ==========================
cv183x_tpu_clock:          650Mhz
tdma_load:               50.28MB, tdma_store:               9.57MB, tdma_total               59.85MB
tdma_exe_tick:          6713403t, tiu_exe_tick            6884902t, inference_tick          10804500t
tdma_exe_percent:          62.14%, tiu_exe_percent:          63.72%, paralellism_percent       125.86%
tdma_exe_ms:             10.33ms, tiu_exe_ms:              10.59ms, inference_ms:             16.62ms


=======================inference total info ==========================
cv183x_tpu_clock:          650Mhz
tdma_load:               50.28MB, tdma_store:               9.57MB, tdma_total               59.85MB
tdma_exe_tick:         13556770t, tiu_exe_tick            6884902t, inference_tick          15720929t
tdma_exe_percent:          86.23%, tiu_exe_percent:          43.79%, paralellism_percent       130.03%
tdma_exe_ms:             20.86ms, tiu_exe_ms:              10.59ms, inference_ms:             24.19ms
```

相关参数说明：

参数	说明
cv18xx_tpu_clock	tpu频率
tdma_total	tpu dma数据搬运总量
tdma_exe_ms	tpu dma数据搬运耗时；此项耗时与内存带宽相关，若内存带宽不够用，耗时可能会增加
tiu_exe_ms	tiu耗时（tpu运算耗时）
inference_ms	推理总耗时
paralellism_percent	tiu与tdma并行率 （tiu_exe_ms + tdma_exe_ms）/ inference_ms

步骤 6：量化模型的选择和部署

在业务场景部署中, 由于bf16量化的模型相似度高, 优先选择步骤三生成的bf16 cvimodel来搭配cvitek_tpu_samples的使用。

在cvitek_tpu_samples中确定代码中的预处理实现与转模型脚本的预处理参数一致, 保证模型推理出来分类或者检测结果正确。

在保证模型推理结果正确下，根据步骤五测试出来模型性能数据和真实场景需求, 选择是否需要使用int8量化模型。

若权衡int8模型的效果比较差以及bf16模型的速度比较慢，可以根据第七章来选择介于bf16和int8之间的混精度量化模型。

模型在开发板上的部署可直接参考本文档的runtime samples最后一章。

4 编译移植caffe模型

本章以mobilenet_v2为例，介绍如何编译迁移一个caffe模型至TPU平台运行

本章需要如下文件：

- cvitek_mlir_ubuntu-18.04.tar.gz

步骤 0：加载cvitek_mlir环境

```
tar zxf cvitek_mlir_ubuntu-18.04.tar.gz
source cvitek_mlir/cvitek_envs.sh
```

步骤 1：获取caffe模型

从https://github.com/shicai/MobileNet-Caffe下载模型，并保存在`model_mobilenet_v2`目录：

```
mkdir model_mobilenet_v2 && cd model_mobilenet_v2
wget -nc https://github.com/shicai/MobileNet-
Caffe/raw/master/mobilenet_v2.caffemodel
wget -nc https://github.com/shicai/MobileNet-
Caffe/raw/master/mobilenet_v2_deploy.prototxt
```

创建工作目录workspace，拷贝测试图片cat.jpg，和数据集100张图片（来自ILSVRC2012）：

```
mkdir workspace && cd workspace
cp $MLIR_PATH/tpuc/regression/data/cat.jpg .
cp -rf $MLIR_PATH/tpuc/regression/data/images .
```

步骤 2：caffe模型转换为fp32 mlir形式

使用`model_transform.py`将模型转换成mlir文件，其中可支持预处理参数如下：

由caffe模型转换为mlir，执行以下shell：

```
model_transform.py \
  --model_type caffe \
  --model_name mobilenet_v2 \
  --model_def ../mobilenet_v2_deploy.prototxt \
  --model_data ../mobilenet_v2.caffemodel \
  --image ./cat.jpg \
  --image_resize_dims 256,256 \
  --keep_aspect_ratio false \
  --net_input_dims 224,224 \
  --raw_scale 255.0 \
  --mean 103.94,115.78,123.68 \
  --std 1.0,1.0,1.0 \
  --input_scale 0.017 \
  --model_channel_order "bgr" \
  --tolerance 0.99,0.99,0.99 \
  --excepts prob \
  --mlir mobilenet_v2_fp32.mlir
```

得到`mobilenet_v2_fp32.mlir`文件.

步骤 3：生成全bf16量化cvimodel

```
model_deploy.py \
  --model_name mobilenet_v2 \
  --mlir mobilenet_v2_fp32.mlir \
  --quantize BF16 \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.94,0.94,0.61 \
  --correctness 0.99,0.99,0.96 \
  --cvimodel mobilenet_v2_bf16.cvimodel
```

步骤 4：生成全int8量化cvimodel

先做Calibration。Calibration前需要先准备校正图片集,图片的数量根据情况准备100~1000张左右。这里用100张图片举例，执行calibration：

```
run_calibration.py \
  mobilenet_v2_fp32.mlir \
  --dataset=./images \
  --input_num=100 \
  -o mobilenet_v2_calibration_table
```

得到`mobilenet_v2_calibration_table`。

导入calibration_table，生成cvimodel：

```
model_deploy.py \
  --model_name mobilenet_v2 \
  --mlir mobilenet_v2_fp32.mlir \
  --calibration_table mobilenet_v2_calibration_table \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.94,0.94,0.61 \
  --correctness 0.99,0.99,0.99 \
  --cvimodel mobilenet_v2_int8.cvimodel
```

步骤 5：开发板中测试bf16和int8模型

配置开发板的TPU sdk环境：

```
# 此处以183x举例
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

测试仿真环境与真实硬件的输出结果，需要步骤三或步骤四生成的调试文件：

- xxx_quantized_tensors_sim.npz 仿真环境中网络推理过程的tensor文件，作为与真实硬件输出结果参考对比
- xxx_in_fp32.npz 模型的输入tensor文件，测试不同类型的模型，input_npz文件需要不一样
- xxx_[int8/bf16].cvimodel 输出int8或者bf16的cvimodel文件

在开发板中执行以下shell，测试量化为int8模型的仿真环境和真实硬件输出结果进行比较：

```
model_runner \
--input mobilenet_v2_in_fp32.npz \
--model mobilenet_v2_int8.cvimodel \
--output out.npz \
--reference mobilenet_v2_quantized_tensors_sim.npz
```

5 多输入模型转换

在docker下cvitek_mlir环境中，用pytorch自定义一个简单的模型，介绍如何转换多输入模型

步骤 0：使用pytorch来构建简易的多输入模型

```
import torch
import torch.nn as nn

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_1d = nn.Conv1d(in_channels=3, out_channels=1, kernel_size=3,
padding=1)
        self.layer_norm = nn.LayerNorm(100)
        self.rnn = nn.LSTM(input_size=100, hidden_size=128, bidirectional=True)
    def forward(self, x, h_0, c_0):
        x = self.conv_1d(x)
        x = self.layer_norm(x)
        Y, (Y_h, Y_c) = self.rnn(x, (h_0, c_0))
        return Y, Y_h, Y_c

net = Net()
inputs = (torch.randn(81, 3, 100), torch.randn(2, 1, 128), torch.randn(2, 1,
128))

torch.onnx.export(net,
      inputs,
     "sample_model.onnx",
      export_params=True,
      opset_version=13,
      input_names=["input",'h_0','c_0'],
      dynamic_axes=None,
      )
```

步骤 1：模拟输入

这里用随机数模拟输入，请以实际模型的输入为准。用一个npz或者用多个npy都可以。如下：

```
import numpy as np
input_data = np.random.rand(81, 3, 100).astype(np.float32)
h0_data = np.random.rand(2, 1, 128).astype(np.float32)
c0_data = np.random.rand(2, 1, 128).astype(np.float32)

# npy文件
np.save("input", input_data)
np.save("h_0", h0_data)
np.save("c_0", c0_data)
# npz文件
np.savez('in_all', input=input_data, h_0=h0_data, c_0=c0_data)
```

步骤 2：onnx模型转换为fp32 mlir形式

支持用一个npz文件作为测试输入

```
model_transform.py \
  --model_type onnx \
  --model_name sample \
  --model_def sample_model.onnx \
  --image in_all.npz \
  --tolerance 0.99,0.99,0.99 \
  --mlir sample_model.mlir
```

也可以支持多个npy形式的输入（可选，后文皆以npz文件为例）

```
model_transform.py \
  --model_type onnx \
  --model_name sample_model \
  --model_def sample_model.onnx \
  --image input.npy,h_0.npy,c_0.npy \
  --tolerance 0.99,0.99,0.99 \
  --mlir sample_model.mlir
```

特别要说明的是，如果输入中既有图片，也有非图片，则可以加入图片的预处理参数，预处理仅对图片生效，输入形式：`--image dog.jpg,h_0.npy,c_0.npy`

步骤 3：生成全bf16量化cvimodel

```
model_deploy.py \
  --model_name sample_model \
  --mlir sample_model.mlir \
  --quantize BF16 \
  --chip cv183x \
  --image in_all.npz \
  --tolerance 0.95,0.95,0.80 \
  --correctness 0.95,0.95,0.90 \
  --cvimodel sample_model.cvimodel
```

步骤 4：生成全int8量化cvimodel

先做Calibration。Calibration前需要先准备少量数据集，一般建议100-1000之间。
这里举例，只用一个npz的数据来执行calibration。首先生成cali_list文件，命令如下：

```
echo in_all.npz > cali_list
```

多个npy文件，也可以如下表示：（可选）

```
echo input.npy,h_0.npy,c_0.npy > cali_list
```

然后执行`run_calibration.py`，如下：

```
run_calibration.py \
  sample_model.mlir \
  --image_list=cali_list \
  --input_num=1 \
  -o sample_calibration_table
```

得到`sample_calibration_table`。

导入calibration_table，生成cvimodel：

```
model_deploy.py \
  --model_name sample_model \
  --mlir sample_model.mlir \
  --calibration_table sample_calibration_table \
  --chip cv183x \
  --image in_all.npz \
  --tolerance 0.9,0.9,0.6 \
  --correctness 0.95,0.95,0.9 \
  --cvimodel sample_model_int8.cvimodel
```

6 算子验证

TPU工具链中包含算子测试用例，test_torch.py和test_onnx.py。

执行方法如下：

```
source cvitek_mlir/cvitek_envs.sh
## 测试torch的LayerNorm
test_torch.py LayerNorm
## 测试torch全部算子用例
test_torch.py
## 测试onnx的Conv2d
test_onnx.py Conv2d
## 测试onnx的全部算子用例
test_onnx.py
```

用户可以参考修改test_torch.py和test_onnx.py，对算子进行测试。

docker中测试完后会生成相应的cvimodel模型和输入文件，可以放入开发板，用于测试算子的性能。

这里取`Add_int8.cvimodel`模型和`Add_in_fp32.npz`输入来进行举例：

```
export TPU_ENABLE_PMU=1
model_runner --input Add_in_fp32.npz --model Add_int8.cvimodel
```

7 精度优化和混合量化

TPU支持INT8和BF16两种量化方法。在模型编译阶段，工具链支持以搜索的方式找到对模型精度最敏感的op，并支持将指定数量的op替换为BF16，从而提高整个网络的精度。

本章以`mobilenet_v1_0.25`为例，介绍如何对这个模型采用自动搜索和混合精度的方式提高模型精度。

此处如何测试精度省略，可以参考各种网络的官网eval脚本，也可以参考`cvitek_mlir`中的`eval_classifier.py`脚本。

本章需要如下文件：

- cvitek_mlir_ubuntu-18.04.tar.gz

步骤 0：获取tensorflow模型，并转换为onnx模型

使用tensorflow提供的`mobilenet_v1_0.25_224`模型，参见：https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

下载链接：http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_0.25_224.tgz

首先打开`mobilenet_v1_0.25_224_eval.pbtxt`，找到输出节点名称为`MobilenetV1/Predictions/Reshape_1`，
使用下列命令转换为onnx模型：

```
source cvitek_mlir/cvitek_envs.sh
pip install tf2onnx
mkdir model_mnet_25 && cd model_mnet_25
wget -nc
http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_0.25_
224.tgz
tar xzf mobilenet_v1_0.25_224.tgz
python3 -m tf2onnx.convert --graphdef mobilenet_v1_0.25_224_frozen.pb \
    --output mnet_25.onnx --inputs input:0 \
    --outputs MobilenetV1/Predictions/Reshape_1:0
```

得到mnet_25.onnx。

但是由于tensorflow模型默认采用NHWC作为输入，转为onnx模型后，仍然是NHWC格式输入，并连接一个transpose节点。在编译前，我们先转换输入格式，并去除这个transpose节点。采用如下python脚本进行：

```
import onnx
model = onnx.load('mnet_25.onnx')
print(model.graph.input[0].type.tensor_type.shape.dim)
model.graph.input[0].type.tensor_type.shape.dim[1].dim_value = 3
model.graph.input[0].type.tensor_type.shape.dim[2].dim_value = 224
model.graph.input[0].type.tensor_type.shape.dim[3].dim_value = 224
print(model.graph.input[0].type.tensor_type.shape.dim)
input_name = model.graph.input[0].name
del model.graph.node[0]
model.graph.node[0].input[0] = input_name
onnx.save(model, 'mnet_25_new.onnx')
```

得到`mnet_25_new.onnx`。

若tensorflow模型为saved_model的pb形式，需要进行转化为frozen_model的pb形式

步骤 1：模型转换为fp32 mlir形式

取得一张测试用图片，本示例使用cvitek_mlir包含的cat.jpg：

```
mkdir workspace && cd workspace
cp $MLIR_PATH/tpuc/regression/data/cat.jpg .
cp -rf $MLIR_PATH/tpuc/regression/data/images .
```

预处理参数如下：

> RAW_SCALE=255
>
> MODEL_CHANNEL_ORDER="rgb"
>
> MEAN=127.5,127.5,127.5 # in RGB
>
> STD=127.5,127.5,127.5
>
> INPUT_SCALE=1.0

转换为mlir文件:

```
model_transform.py \
  --model_type onnx \
  --model_name mnet_25 \
  --model_def ../mnet_25_new.onnx \
  --image ./cat.jpg \
  --image_resize_dims 256,256 \
  --keep_aspect_ratio false \
  --net_input_dims 224,224 \
  --raw_scale 255.0 \
  --mean 127.5,127.5,127.5 \
  --std 127.5,127.5,127.5 \
  --input_scale 1.0 \
  --model_channel_order "rgb" \
  --tolerance 0.99,0.99,0.99 \
  --mlir mnet_25_fp32.mlir
```

得到`mnet_25_fp32.mlir`文件。

使用ILSVRC2012数据集验证精度，测试得到FP32模型精度为Top-1 49.2% Top-5 73.5%。

步骤 2：进行INT8量化

进行calibration：

```
run_calibration.py \
    mnet_25_fp32.mlir \
    --dataset=./images \
    --input_num=100 \
    --calibration_table mnet_25_calibration_table
```

得到`mnet_25_calibration_table`。

进行INT8量化，并进行逐层比较：

```
model_deploy.py \
  --model_name mnet_25 \
  --mlir mnet_25_fp32.mlir \
  --calibration_table mnet_25_calibration_table \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.93,0.90,0.60 \
  --correctness 0.99,0.99,0.99 \
  --cvimodel mnet_25.cvimodel
```

测试得到INT8模型精度为Top-1 43.2% Top-5 68.3%，比FP32模型精度有一定幅度下降。

步骤 3：进行BF16量化

```
model_deploy.py \
  --model_name mnet_25 \
  --mlir mnet_25_fp32.mlir \
  --quantize BF16 \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.99,0.99,0.86 \
  --correctness 0.99,0.99,0.93 \
  --cvimodel mnet_25_all_bf16_precision.cvimodel
```

测试BF16模型精度为Top-1 48.49 Top-5 73.064。

步骤 4：进行混合量化搜索，并进行混合量化

搜索混合量化表。此模型共有59层，选择多少层进行替换，可以根据对精度的需要，以及测试的精度结果来进行调整。搜索用的数据集数量也可以根据需要调整。

此处以替换其中6层为例（`--max_bf16_layers=6`），搜索用的测试数据集为100张：

```
run_mix_precision.py \
    mnet_25_fp32.mlir \
    --dataset ./images \
    --input_num=20 \
    --calibration_table mnet_25_calibration_table \
    --max_bf16_layers=6 \
    -o mnet_25_mix_precision_bf16_table
```

得到`mnet_25_mix_precision_bf16_table`，内容如下：

```
cat mnet_25_mix_precision_bf16_table
# MobilenetV1/MobilenetV1/Conv2d_2_depthwise/Relu6:0_relu6_reluClip
# MobilenetV1/MobilenetV1/Conv2d_1_depthwise/Relu6:0_relu6_reluClip
# MobilenetV1/MobilenetV1/Conv2d_1_pointwise/Relu6:0_relu6_reluClip
# MobilenetV1/MobilenetV1/Conv2d_1_depthwise/Relu6:0_Clip
# MobilenetV1/MobilenetV1/Conv2d_0/Relu6:0_Clip
# MobilenetV1/MobilenetV1/Conv2d_0/Relu6:0_relu6_reluClip
```

进行混合量化：

```
model_deploy.py \
  --model_name mnet_25 \
  --mlir mnet_25_fp32.mlir \
  --calibration_table mnet_25_calibration_table \
  --mix_precision_table mnet_25_mix_precision_bf16_table \
  --quantize INT8 \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.94,0.93,0.67 \
  --correctness 0.99,0.99,0.99 \
  --cvimodel mnet_25_mix_precision.cvimodel
```

测试混合量化模型精度为Top-1 47.4% Top-5 72.3%。

为比较效果，我们调整number_bf16参数分别为10和15，测试混和量化精度。最终结果如下：

Quant Type	Top-1	Top-5
INT8	43.2%	68.3%
MIXED (6 layers)	47.4%	72.3%
MIXED (10 layers)	47.6%	72.3%
MIXED (15 layers)	47.8%	72.5%
BF16	48.5%	73.1%
FP32	49.2%	73.5%

混精度规则说明

规则一： quantize参数可以指定默认的量化方式

目前支持三种量化方式，BF16、INT8、MIX_BF16。其中MIX_BF16方式，精度和性能都介于BF16与INT8之间，ION内存占用与INT8相同。

规则二： mix_precision_table指定特定量化方式，优先级高于quantize

mix_precision_table文件格式如下：

```
# layer_name quantize_type
# 如果没有跟quantize_type，则认为是BF16
120_Add BF16
121_Conv INT8
122_Conv
```

规则三： 当存在INT8的量化时，需要传入calibration_table

当quantize指定为INT8时，或者mix_precision_table中存在layer被指定为INT8时，则需要传入calibration参数。

8 使用TPU做前处理

TPU可以用于支持常见的前处理计算，包括raw_scale，mean，input_scale，channel swap，split，以及quantization。开编发者可以在模型编译阶段，通过译选项传递相应预处理参数，由编译器直接在模型运输前插入相应前处理算子，生成的cvimodel即可以直接以预处理前的图像作为输入，随模型推理过程使用TPU处理前处理运算。

本章介绍使用TPU做前处理的具体步骤。本章以Caffe模型编译为例，按照第6章的步骤稍做修改，生成支持前处理的cvimodel。以`mobilenet_v2`为例。

步骤 0-4：与Caffe章节相应步骤相同

假设用户以及按照第4章步骤0到步骤2执行完模型转换后, 并且跳过步骤3全量化为bf16模型，到步骤4生成全INT8量化模型

步骤 5：模型量化并生成含TPU预处理的cvimodel

首先，加载cvitek_mlir环境：

```
source cvitek_mlir/cvitek_envs.sh
cd model_mobilenet_v2/workspace
```

执行以下命令：

```
#与预处理有关的参数：fuse_preprocess/pixel_format/aligned_input
model_deploy.py \
  --model_name mobilenet_v2 \
  --mlir mobilenet_v2_fp32.mlir \
  --calibration_table mobilenet_v2_calibration_table \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.94,0.94,0.61 \
  --fuse_preprocess \
  --pixel_format BGR_PACKED \
  --aligned_input false \
  --excepts data \
  --correctness 0.99,0.99,0.99 \
  --cvimodel mobilenet_v2_fused_preprocess.cvimodel
```

就可以得到带前处理的cvimodel.

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。

以上过程包含以下几步:

- 生成带前处理的MLIR int8模型, 以及不包含前处理的输入 mobilenet_v2_resized_only_in_fp32.npz
- 执行MLIR int8推理 与 MLIR fp32 推理结果的比较, 验证MLIR int8 带前处理模型的正确性
- 生成带前处理的 cvimodel, 以及调用仿真器执行推理, 将结果与 MLIR int8 带前处理的模型的推理结果做比较

步骤 6：开发板中测试带前处理的模型

配置开发板的TPU sdk环境

```
# 183x为例
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

测试仿真环境与真实硬件的输出结果，需要之前步骤生成的调试文件：

- xxx_quantized_tensors_sim.npz 仿真环境中网络推理过程的tensor文件，作为与真实硬件输出结果参考对比
- xxx_[int8/bf16].cvimodel　　输出int8或者bf16的cvimodel文件
- xxx_in_fp32_resize_only.npz　未经过前处理的输入数据文件

在开发板中执行以下shell，测试量化为int8模型的仿真环境和真实硬件输出结果进行比较

```
model_runner \
--input mobilenet_v2_in_fp32_resize_only.npz \
--model mobilenet_v2_fused_preprocess.cvimodel \
--output out.npz --dump-all-tensors \
--reference mobilenet_v2_quantized_tensors_sim.npz
```

以上过程包含将为经过未前处理的输入文件放到带前处理的cvi_model模型中推理，并且将输出的结果和在仿真环境输出的结果进行比较，保证误差在可控范围内。

使用VPSS Frame作为模型输入

VPSS是CV18xx提供的视频处理模块，视频处理流水线输出的图像数据，可以直接用做神经网络输入数据。VPSS的详细使用方法请参阅《CV18xx 媒体软件开发参考》，本文档不做介绍。使用VPSS frame作为模型输入必须使用TPU做前处理，操作步骤同上，仅需将**--aligned_input false改为true**，原因是VPSS的图片数据是有对齐要求的，因此如果要实现无拷贝将VPSS的图片输入到模型就需要模型遵循VPSS的对齐格式去读取input数据。

9 合并cvimodel模型文件

对于同一个模型，可以依据输入的batch size以及分辨率(不同的h和w)分别生成独立的cvimodel文件。不过为了节省外存和运存，可以选择将这些相关的cvimodel文件合并为一个cvimodel文件，共享其权重部分。具体步骤如下：

步骤 0：生成batch 1的cvimodel

请参考前述章节，新建workspace，通过model_transform.py将mobilenet_v2的caffemodel转换为mlir fp32模型:

重要:

```
1. 如沿用之前的workspace需清空workspace；
2. 步骤0、步骤1 中 --merge_weigh 为必需选项。
```

```
model_transform.py \
  --model_type caffe \
  --model_name mobilenet_v2 \
  --model_def ../mobilenet_v2_deploy.prototxt \
  --model_data ../mobilenet_v2.caffemodel \
  --image ./cat.jpg \
  --image_resize_dims 256,256 \
  --keep_aspect_ratio false \
  --net_input_dims 224,224 \
  --raw_scale 255.0 \
  --mean 103.94,115.78,123.68 \
  --std 1.0,1.0,1.0 \
  --input_scale 0.017 \
  --model_channel_order "bgr" \
  --batch_size 1 \
  --tolerance 0.99,0.99,0.99 \
  --excepts prob \
  --mlir mobilenet_v2_fp32_bs1.mlir
```

使用第4章节生成的`mobilenet_v2_calibration_table`；如果没有，则通过run_calibration.py工具对mobilenet_v2_fp32.mlir进行量化校验获得calibration table文件.

然后将模型量化并生成cvimodel：

```
# 加上--merge_weight 参数
model_deploy.py \
  --model_name mobilenet_v2 \
  --mlir mobilenet_v2_fp32_bs1.mlir \
  --calibration_table mobilenet_v2_calibration_table \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.95,0.94,0.69 \
  --correctness 0.99,0.99,0.99 \
  --merge_weight  \
  --cvimodel mobilenet_v2_bs1.cvimodel
```

步骤 1：生成batch 4的cvimodel

同步骤1，在同一个workspace中生成batch为4的mlir fp32文件:

```
model_transform.py \
  --model_type caffe \
  --model_name mobilenet_v2 \
  --model_def ../mobilenet_v2_deploy.prototxt \
  --model_data ../mobilenet_v2.caffemodel \
  --image ./cat.jpg \
  --image_resize_dims 256,256 \
  --keep_aspect_ratio false \
  --net_input_dims 224,224 \
  --raw_scale 255.0 \
  --mean 103.94,115.78,123.68 \
  --std 1.0,1.0,1.0 \
  --input_scale 0.017 \
  --model_channel_order "bgr" \
  --batch_size 4 \
  --tolerance 0.99,0.99,0.99 \
  --excepts prob \
  --mlir mobilenet_v2_fp32_bs4.mlir
```

使用`mobilenet_v2_calibration_table`文件将模型量化并生成cvimodel:

```
 # 打开--merge_weight选项
 model_deploy.py \
  --model_name mobilenet_v2 \
  --mlir mobilenet_v2_fp32_bs4.mlir \
  --calibration_table mobilenet_v2_calibration_table \
  --chip cv183x \
  --image cat.jpg \
  --tolerance 0.95,0.94,0.69 \
  --correctness 0.99,0.99,0.99 \
  --merge_weight \
  --cvimodel mobilenet_v2_bs4.cvimodel
```

步骤 2: 合并batch 1和batch 4的cvimodel

使用cvimodel_tool合并两个cvimodel文件:

```
 cvimodel_tool \
  -a merge \
  -i mobilenet_v2_bs1.cvimodel \
     mobilenet_v2_bs4.cvimodel \
  -o mobilenet_v2_bs1_bs4.cvimodel
```

步骤 3：runtime接口调用cvimodel

在运行时可以通过命令：

```
cvimodel_tool -a dump -i mobilenet_v2_bs1_bs4.cvimodel
```

查看bs1和bs4指令的program id，在运行时可以透过如下方式去运行不同的batch指令：

```
CVI_MODEL_HANDEL bs1_handle;
CVI_RC ret = CVI_NN_RegisterModel("mobilenet_v2_bs1_bs4.cvimodel", &bs1_handle);
assert(ret == CVI_RC_SUCCESS);
CVI_NN_SetConfig(bs1_handle, OPTION_PROGRAM_INDEX, 0);
CVI_NN_GetInputOutputTensors(bs_handle, ...);
....


CVI_MODEL_HANDLE bs4_handle;
// 复用已加载的模型
CVI_RC ret = CVI_NN_CloneModel(bs1_handle, &bs4_handle);
assert(ret == CVI_RC_SUCCESS);
// 选择bs4的指令
CVI_NN_SetConfig(bs4_handle, OPTION_PROGRAM_INDEX, 1);
CVI_NN_GetInputOutputTensors(bs_handle, ...);
...

// 最后销毁bs1_handle, bs4_handel
CVI_NN_CleanupModel(bs1_handle);
CVI_NN_CleanupModel(bs4_handle);
```

综述：合并过程

使用上面的方面，不论是相同模型还是不同模型，均可以进行合并。
合并的原理是：模型生成过程中，会叠加前面模型的weight（如果相同则共用）。
主要步骤在于：

1. 用`model_deploy.py`生成模型时，加上`--merge_weight`参数
2. 要合并的模型的生成目录必须是同一个，且在合并模型前不要清理任何中间文件
3. 用`cvimodel_tool -a merge`将多个cvimodel合并

10 运行runtime samples

本章首先介绍EVB如何运行sample应用程序，然后介绍如何交叉编译sample应用程序，最后介绍docker仿真编译和运行sample。具体包括3个samples：

- Sample-1 : classifier (mobilenet_v2)
- Sample-2 : classifier_bf16 (mobilenet_v2)
- Sample-3 : classifier fused preprocess (mobilenet_v2)
- Sample-4 : classifier multiple batch (mobilenet_v2)

1) EVB运行Samples程序

在EVB运行release提供的sample预编译程序。

需要如下文件：

- cvitek_tpu_sdk_[cv182x|cv182x_uclibc|cv183x|cv181x_glibc32|cv181x_musl_riscv64].tar.gz
- cvimodel_samples_[cv182x|cv183x|cv181x].tar.gz

将根据chip类型选择所需文件加载至EVB的文件系统，于evb上的linux console执行，以cv183x为例：

解压samples使用的model文件（以cvimodel格式交付），并解压TPU_SDK，并进入samples目录，执行测试，过程如下：

```
# envs
tar zxf cvimodel_samples_cv183x.tar.gz
export MODEL_PATH=$PWD/cvimodel_samples
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export TPU_ROOT=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh

# get cvimodel info
cd samples
./bin/cvi_sample_model_info $MODEL_PATH/mobilenet_v2.cvimodel

#################################
# sample-1 : classifier
#################################
./bin/cvi_sample_classifier \
    $MODEL_PATH/mobilenet_v2.cvimodel \
    ./data/cat.jpg \
    ./data/synset_words.txt

# TOP_K[5]:
#  0.326172, idx 282, n02123159 tiger cat
#  0.326172, idx 285, n02124075 Egyptian cat
#  0.099609, idx 281, n02123045 tabby, tabby cat
#  0.071777, idx 287, n02127052 lynx, catamount
#  0.041504, idx 331, n02326432 hare

####################################
# sample-2 : classifier_bf16
####################################
./bin/cvi_sample_classifier_bf16 \
    $MODEL_PATH/mobilenet_v2_bf16.cvimodel \
    ./data/cat.jpg \
    ./data/synset_words.txt
```

```
# TOP_K[5]:
#  0.314453, idx 285, n02124075 Egyptian cat
#  0.040039, idx 331, n02326432 hare
#  0.018677, idx 330, n02325366 wood rabbit, cottontail, cottontail rabbit
#  0.010986, idx 463, n02909870 bucket, pail
#  0.010986, idx 852, n04409515 tennis ball


##########################################
# sample-3 : classifier fused preprocess
##########################################
./bin/cvi_sample_classifier_fused_preprocess \
    $MODEL_PATH/mobilenet_v2_fused_preprocess.cvimodel \
    ./data/cat.jpg \
    ./data/synset_words.txt

# TOP_K[5]:
#  0.326172, idx 282, n02123159 tiger cat
#  0.326172, idx 285, n02124075 Egyptian cat
#  0.099609, idx 281, n02123045 tabby, tabby cat
#  0.071777, idx 287, n02127052 lynx, catamount
#  0.041504, idx 331, n02326432 hare

##########################################
# sample-4 : classifier multiple batch
##########################################
./bin/cvi_sample_classifier_multi_batch \
    $MODEL_PATH/mobilenet_v2_bs1_bs4.cvimodel \
    ./data/cat.jpg \
    ./data/synset_words.txt

# TOP_K[5]:
#  0.326172, idx 282, n02123159 tiger cat
#  0.326172, idx 285, n02124075 Egyptian cat
#  0.099609, idx 281, n02123045 tabby, tabby cat
#  0.071777, idx 287, n02127052 lynx, catamount
#  0.041504, idx 331, n02326432 hare
```

同时提供脚本作为参考，执行效果与直接运行相同，如下：

```
./run_classifier.sh
./run_classifier_bf16.sh
./run_classifier_fused_preprocess.sh
./run_classifier_multi_batch.sh
```

在cvitek_tpu_sdk/samples/samples_extra目录下有更多的samples，可供参考：

```
./bin/cvi_sample_detector_yolo_v3_fused_preprocess \
    $MODEL_PATH/yolo_v3_416_fused_preprocess_with_detection.cvimodel \
    ./data/dog.jpg \
    yolo_v3_out.jpg

./bin/cvi_sample_detector_yolo_v5_fused_preprocess \
```

```
    $MODEL_PATH/yolov5s_fused_preprocess.cvimodel \
    ./data/dog.jpg \
    yolo_v5_out.jpg

./bin/cvi_sample_detector_yolox_s \
    $MODEL_PATH/yolox_s.cvimodel \
    ./data/dog.jpg \
    yolox_s_out.jpg

./bin/cvi_sample_alphapose_fused_preprocess \
    $MODEL_PATH/yolo_v3_416_fused_preprocess_with_detection.cvimodel \
    $MODEL_PATH/alphapose_fused_preprocess.cvimodel \
    ./data/pose_demo_2.jpg \
    alphapose_out.jpg

./bin/cvi_sample_fd_fr_fused_preprocess \
    $MODEL_PATH/retinaface_mnet25_600_fused_preprocess_with_detection.cvimodel \
    $MODEL_PATH/arcface_res50_fused_preprocess.cvimodel \
    ./data/obama1.jpg \
    ./data/obama2.jpg
```

2) 交叉编译samples程序

发布包有samples的源代码，按照本节方法在Docker环境下交叉编译samples程序，然后在evb上运行。

本节需要如下文件：

- cvitek_tpu_sdk_[cv182x|cv182x_uclibc|cv183x|cv181x_glibc32|cv181x_musl_riscv64].tar.gz
- cvitek_tpu_samples.tar.gz

64位平台

> 如cv183x 64位平台

TPU sdk准备：

```
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

编译samples，安装至install_samples目录：

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_C_FLAGS_RELEASE=-O3 -DCMAKE_CXX_FLAGS_RELEASE=-O3 \
    -DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-aarch64-linux.cmake \
    -DTPU_SDK_PATH=$TPU_SDK_PATH \
    -DOPENCV_PATH=$TPU_SDK_PATH/opencv \
    -DCMAKE_INSTALL_PREFIX=../install_samples \
    ..
cmake --build . --target install
```

32位平台

> 如cv183x平台32位、cv182x、cv181x_glibc32

TPU sdk准备：

```
tar zxf cvitek_tpu_sdk_cv182x.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

如果docker版本低于1.7，则需要更新32位系统库（只需一次）：

```
dpkg --add-architecture i386
apt-get update
apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

编译samples，安装至install_samples目录：

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_C_FLAGS_RELEASE=-O3 -DCMAKE_CXX_FLAGS_RELEASE=-O3 \
    -DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-linux-gnueabihf.cmake \
    -DTPU_SDK_PATH=$TPU_SDK_PATH \
    -DOPENCV_PATH=$TPU_SDK_PATH/opencv \
    -DCMAKE_INSTALL_PREFIX=../install_samples \
    ..
cmake --build . --target install
```

uclibc 32位平台

> cv182x 32位平台 uclibc

TPU sdk准备：

```
tar zxf cvitek_tpu_sdk_cv182x_uclibc.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

如果docker版本低于1.7，则需要更新32位系统库（只需一次）：

```
dpkg --add-architecture i386
apt-get update
apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

编译samples，安装至install_samples目录：

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_C_FLAGS_RELEASE=-O3 -DCMAKE_CXX_FLAGS_RELEASE=-O3 \
    -DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-linux-uclibc.cmake \
    -DTPU_SDK_PATH=$TPU_SDK_PATH \
    -DOPENCV_PATH=$TPU_SDK_PATH/opencv \
    -DCMAKE_INSTALL_PREFIX=../install_samples \
    ..
cmake --build . --target install
```

cv181x riscv64位平台

> cv181x riscv64位平台

TPU sdk准备：

```
tar zxf cvitek_tpu_sdk_cv181x_musl_riscv64.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

如果docker版本低于1.7，则需要更新32位系统库（只需一次）：

```
dpkg --add-architecture i386
apt-get update
apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

编译samples，安装至install_samples目录：

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_C_FLAGS_RELEASE=-O3 -DCMAKE_CXX_FLAGS_RELEASE=-O3 \
    -DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-riscv64-linux-musl-
x86_64.cmake \
    -DTPU_SDK_PATH=$TPU_SDK_PATH \
    -DOPENCV_PATH=$TPU_SDK_PATH/opencv \
    -DCMAKE_INSTALL_PREFIX=../install_samples \
    ..
cmake --build . --target install
```

####

3) docker环境仿真运行的samples程序

需要如下文件：

- cvitek_mlir_ubuntu-18.04.tar.gz
- cvimodel_samples_[cv182x|cv183x|cv181x].tar.gz
- cvitek_tpu_samples.tar.gz

TPU sdk准备：

```
tar zxf cvitek_mlir_ubuntu-18.04.tar.gz
source cvitek_mlir/cvitek_envs.sh
```

编译samples，安装至install_samples目录：

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build
cd build
cmake -G Ninja \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_C_FLAGS_RELEASE=-O3 -DCMAKE_CXX_FLAGS_RELEASE=-O3 \
    -DTPU_SDK_PATH=$MLIR_PATH/tpuc \
    -DCNPY_PATH=$MLIR_PATH/cnpy \
    -DOPENCV_PATH=$MLIR_PATH/opencv \
    -DCMAKE_INSTALL_PREFIX=../install_samples \
    ..
cmake --build . --target install
```

运行samples程序：

```
# envs
tar zxf cvimodel_samples_cv183x.tar.gz
export MODEL_PATH=$PWD/cvimodel_samples
source cvitek_mlir/cvitek_envs.sh

# get cvimodel info
cd ../install_samples
./bin/cvi_sample_model_info $MODEL_PATH/mobilenet_v2.cvimodel
```

其他samples运行命令参照EVB运行命令