一、Python Caffe MNIST INT8 量化

TensorRT 对于 Int8 的量化，需要进行校准（calibration），生成校准文件。

1、功能概述：以 Caffe 模型作为输入，通过 MNIST 数据集构建标定所需参数，实现 INT8 量化。

与普通 Caffe 模型转换为 TensorRT Engine 相比，INT8 量化多了以下操作：

在使用 Parser 解析 Caffe 模型的时候，需要指定 weights 输入类型为 FP32；

IBuilderConfig 需要设置 FLAG 为 BuilderFlag::kINT8；

构建标定对象，即 calibrator，作为 IBuilderConfig 的一部分。

2、如何构建标定对象：

最终目标就是构建一个 IInt8Calibrator 对象作为 IBuilderConfig 的输入。

IInt8Calibrator 只是一个抽象类，关键函数就是：

getBatchSize：调用一次，获取标定过程中的 batch size；

bool getBatch(void* bindings[], const char* names[], int nbBindings)：调用多次，获取标定过程中的输入，直到返回 false；

前两个参数的长度相同，且参数一一对应；

第三个参数的数值就是前两个参数的长度；

需要将读取到数据对应的 GPU 地址保存到 bindings 中；

writeCalibrationCache：将标定结果写入本地，在 buidler 运行过程中会自动调用；

readCalibrationCache：读取保存在本地的标定文件，在 buidler 运行过程中会自动调用。

IInt8Calibrator 有具体实现，最常用的是 IInt8EntropyCalibrator2。

builder 如何使用 calibrator：

先调用 getBatchSize 函数，获取标定过程中的 batch size；

多次调用 getBatch 函数，每次获取的参数量就是 getBatchSize 的数值，直到返回 false 为止。

3、构建 INT8 engine 的过程：

构建 FP32 engine，在标定数据集上前向推理，获得每一层结果的直方图；

构建标定表，即 calibration table；

通过 network 以及标定表构建 INT8 engine。

不同硬件平台可以使用相同的标定表（只要网络以及标定数据集没有变化）。

4、不使用 calibrator，而是用户自定义每一层的 INT8 转换数值范围：

功能详解：

使用 nvinfer1::ITensor::setDynamicRange 设置参数的动态范围

使用 nvinfer1::ILayer::setPrecision 设置计算精度，可能是中间过程的参数类型

使用 nvinfer1::ILayer::setOutputType 设置输出数据类型

不使用 INT8 calibration 进行 INT8 量化推理

实现流程：

确定硬件平台支持 INT8 量化（只有 compute capability 6.1 or 7.x 的才支持）

builder 中设置 INT8 模式，且 calibrator 设置为 nullptr

通过 tensor->setDynamicRange(min, max) 来设置动态数值范围

通过 layer->setPrecision(nvinfer1::DataType::kINT8) 来设置精度，中间过程的计算类型

通过 layer->setOutputType(j, nvinfer1::DataType::kFLOAT) 设置输出数据类型，这个应该是模型结果

5、python/int8_caffe_mnist - INT8 Calibration In Python

caliborator 的构建，即 MNISTEntropyCalibrator：

该类是 trt.IInt8EntropyCalibrator2 的子类

也是重写四个函数，get_batch_size/get_batch/read_calibration_cache/write_calibration_cache 上述四个函数的名称与参数可能与 C++版本有少量区别

构建 IBuilder 的时候需要设置 INT8 量化 flag 以及导入 caliborator

校准表应该是在构建 engine 的这一步中生成的。

6、构建 engine：

```python
# 构建engine
# 其中calib是MNISTEntropyCalibrator对象
# This function builds an engine from a Caffe model.
def build_int8_engine(deploy_file, model_file, calib, batch_size=32):
    with trt.Builder(TRT_LOGGER) as builder, builder.create_network() as network, trt.CaffeParser() as parser:
        # We set the builder batch size to be the same as the calibrator's, as we use the same batches
        # during inference. Note that this is not required in general, and inference batch size is
        # independent of calibration batch size.
        builder.max_batch_size = batch_size
        builder.max_workspace_size = common.GiB(1)
        builder.int8_mode = True
        builder.int8_calibrator = calib
        # Parse Caffe model
        model_tensors = parser.parse(deploy=deploy_file, model=model_file, network=network, dtype=ModelData.DTYPE)
        network.mark_output(model_tensors.find(ModelData.OUTPUT_NAME))
        # Build engine and do int8 calibration.
        return builder.build_cuda_engine(network)
```

MNISTEntropyCalibrator：

```python
class MNISTEntropyCalibrator(trt.IInt8EntropyCalibrator2):
    def __init__(self, training_data, cache_file, batch_size=64):
        # Whenever you specify a custom constructor for a TensorRT class,
        # you MUST call the constructor of the parent explicitly.
        trt.IInt8EntropyCalibrator2.__init__(self)

        self.cache_file = cache_file

        # Every time get_batch is called, the next batch of size batch_size will be copied to the device and returned.
        self.data = load_mnist_data(training_data)
        self.batch_size = batch_size
        self.current_index = 0

        # Allocate enough memory for a whole batch.
        self.device_input = cuda.mem_alloc(self.data[0].nbytes * self.batch_size)

    def get_batch_size(self):
        return self.batch_size

    # TensorRT passes along the names of the engine bindings to the get_batch function.
    # You don't necessarily have to use them, but they can be useful to understand the order of
    # the inputs. The bindings list is expected to have the same ordering as 'names'.
    def get_batch(self, names):
        if self.current_index + self.batch_size > self.data.shape[0]:
            return None

        current_batch = int(self.current_index / self.batch_size)
        if current_batch % 10 == 0:
            print("Calibrating batch {:}, containing {:} images".format(current_batch, self.batch_size))

        batch = self.data[self.current_index:self.current_index + self.batch_size].ravel()
        cuda.memcpy_htod(self.device_input, batch)
        self.current_index += self.batch_size
        return [self.device_input]

    def read_calibration_cache(self):
        # If there is a cache, use it instead of calibrating again. Otherwise, implicitly return None.
        if os.path.exists(self.cache_file):
            with open(self.cache_file, "rb") as f:
                return f.read()

    def write_calibration_cache(self, cache):
        with open(self.cache_file, "wb") as f:
            f.write(cache)
```

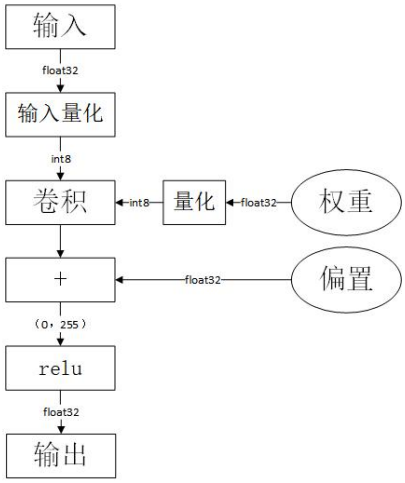
7、输出结果

```
(mytensorrt) zhaohuazheng@user-Super-Server:/data/zhaohuazheng/tensorrt/TensorRT-7.0.0.11/samples/python/int8_caffe_mnist$ python sample.py -d /data/zhaohuazhen
g/tensorrt/TensorRT-7.0.0.11/data/mnist/
WARNING: /usr/src/tensorrt/data/mnist does not exist. Trying /usr/src/tensorrt/data instead.
WARNING: /usr/src/tensorrt/data does not exist. Please provide the correct data path with the -d option.
WARNING: /data/zhaohuazheng/tensorrt/TensorRT-7.0.0.11/data/mnist/mnist does not exist. Trying /data/zhaohuazheng/tensorrt/TensorRT-7.0.0.11/data/mnist/ instead
.
[TensorRT] WARNING: Current optimization profile is: 0. Please ensure there are no enqueued operations pending in this context prior to switching profiles
Calibrating batch 0, containing 64 images
Calibrating batch 10, containing 64 images
Calibrating batch 20, containing 64 images
Calibrating batch 30, containing 64 images
Calibrating batch 40, containing 64 images
Calibrating batch 50, containing 64 images
Calibrating batch 60, containing 64 images
Calibrating batch 70, containing 64 images
Calibrating batch 80, containing 64 images
Calibrating batch 90, containing 64 images
Calibrating batch 100, containing 64 images
Calibrating batch 110, containing 64 images
Calibrating batch 120, containing 64 images
Calibrating batch 130, containing 64 images
Calibrating batch 140, containing 64 images
Calibrating batch 150, containing 64 images
[TensorRT] WARNING: Current optimization profile is: 0. Please ensure there are no enqueued operations pending in this context prior to switching profiles
Validating batch 10
Validating batch 20
Validating batch 30
Validating batch 40
Validating batch 50
Validating batch 60
Validating batch 70
Validating batch 80
Validating batch 90
Validating batch 100
Validating batch 110
Validating batch 120
Validating batch 130
Validating batch 140
Validating batch 150
Validating batch 160
Validating batch 170
Validating batch 180
Validating batch 190
Validating batch 200
Validating batch 210
Validating batch 220
Validating batch 230
Validating batch 240
Validating batch 250
Validating batch 260
Validating batch 270
Validating batch 280
Validating batch 290
Validating batch 300
Validating batch 310
Total Accuracy: 99.09%
(mytensorrt) zhaohuazheng@user-Super-Server:/data/zhaohuazheng/tensorrt/TensorRT-7.0.0.11/samples/python/int8_caffe_mnist$
```

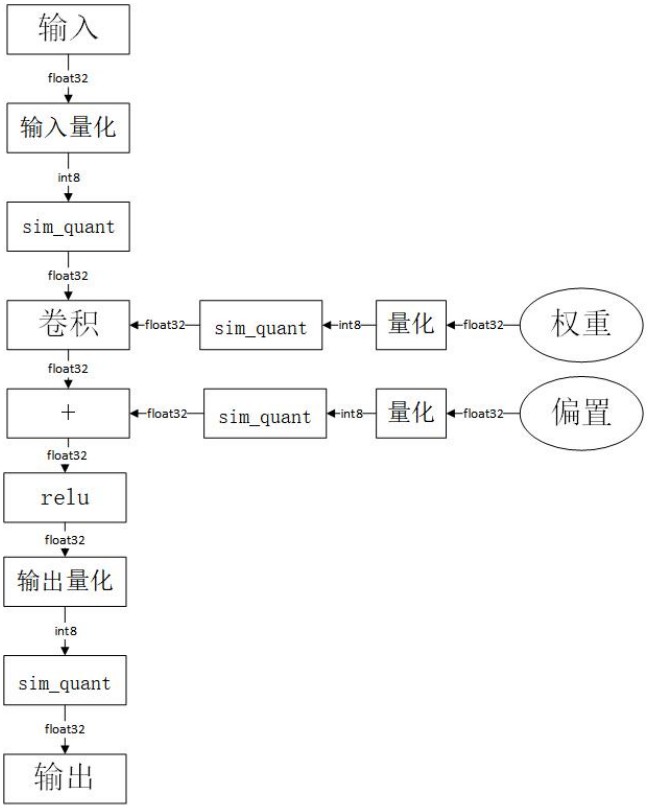
TensorRT 在 mnist 数据集上测试，量化后准确率为 99.09%

二、量化验证

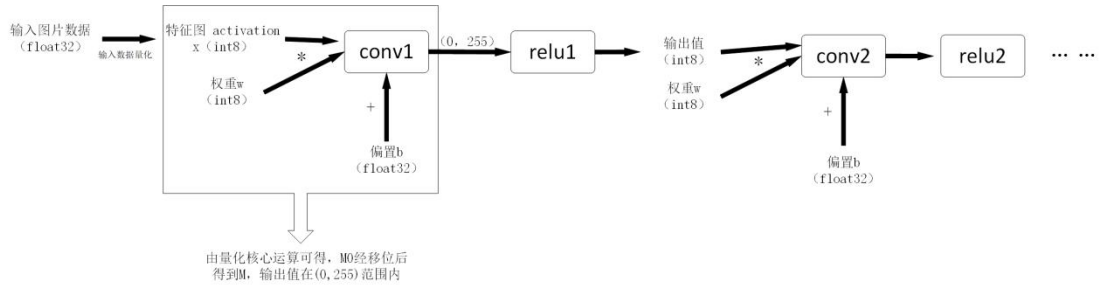
2.1 神经网络量化过程：



2.2 反量化验证过程：



2.3 网络中的量化数据流变化：



2.4 尝试按照他人复现的 tf2 环境下的 yolov3 代码，复现生成 yolov3-tiny（排除之前 tf 版本不兼容的问题）
yolo 网络较为复杂，复现比较有难度.

2.5 在 github 上找到了他人复现生成的 tf2 环境下的 yolov3-tiny，尝试进行量化
量化时遇到的问题： tflite_model = converter.convert()报错

```
tensorflow.lite.python.convert.ConverterError: /home/zhaohuazheng/anaconda3/lib/python3.8/site-packages/tensorflow/python/ops/tensor_array_ops.py:
464:0: error: 'tf.TensorListReserve' op requires element_shape to be 1D tensor during TF Lite transformation pass
```

问题分析：
使用 yolo3-tiny 的预训练权重文件进行量化是没有问题的，但在进行训练之后再进行量化就会有网络模型报错的问题，考虑是 yolov3-tiny 网络复现是有问题的，和官方提供的预训练文件有差异。