华为昇腾-昇思MindSpore《AI计算系统》精品课程

HCIA-AI计算系统

精品课程

实验指导手册

版本:1.0



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[7 MindSpore推理的量化实验 4](#_Toc905453291)

[7.1实验介绍 4](#_Toc884339195)

[7.1.1 数据集介绍 4](#_Toc354529478)

[7.1.2 模型知识点的介绍 4](#_Toc1457299968)

[7.2 实验环境要求 6](#_Toc789568141)

[7.3 实验详细设计与实现 6](#_Toc970290974)

[7.3.1 实验步骤 6](#_Toc1837068347)

[7.3.2 实验流程 22](#_Toc1235315110)

[7.3.3 实验运行 25](#_Toc69154574)

[7.4 实验总结 26](#_Toc492272191)

[7.5 实验任务与参考解答任务 26](#_Toc1511705893)

[7.5.1 实验任务 26](#_Toc361915994)

[7.5.2 参考答案 26](#_Toc1048422854)

# MindSpore推理的量化实验

## 7.1实验介绍

本实验的目的是了解神经网络量化操作，能够独立实现 int8 量化操作，构建量化 VGG16 神经网络，并基于MindSpore框架实现量化推理，能够独立编写量化操作代码。

### 数据集介绍

测试数据集仍然延续实验三四的花卉数据集（雏菊、蒲公英、玫瑰、向日葵、郁金香）。

目录结构如下：

flower\_photos\_test

├── daisy

├── dandelion

├── roses

├── sunflowers

├── tulips

├── LICENSE.txt

### 模型知识点的介绍

7.1.2.1 量化

量化即以较低的推理精度损失将连续取值（或者大量可能的离散取值）的浮点型模型权重或流经模型的张量数据定点近似（通常为INT8）为有限多个（或较少的）离散值的过程，它是以更少位数的数据类型用于近似表示32位有限范围浮点型数据的过程，而模型的输入输出依然是浮点型。这样的好处是可以减小模型尺寸大小，减少模型内存占用，加快模型推理速度，降低功耗等。

如上所述，与FP32类型相比，FP16、INT8、INT4等低精度数据表达类型所占用空间更小。使用低精度数据表达类型替换高精度数据表达类型，可以大幅降低存储空间和传输时间。而低比特的计算性能也更高，INT8相对比FP32的加速比可达到3倍甚至更高，对于相同的计算，功耗上也有明显优势。

当前业界量化方案主要分为两种：感知量化训练（Quantization Aware Training）和训练后量化（Post-training Quantization）。

7.1.2.2 训练后量化

对于已经训练好的float32模型，通过训练后量化将其转为int8，不仅能减小模型大小，而且能显著提高推理性能。训练后量化分为两类：

1. 权重量化：对模型的权值进行量化，仅压缩模型大小，推理时仍然执行float32 推理；

2. 全量化：对模型的权值、激活值等统一进行量化，推理时执行int运算，能提升模型推理速度、降低功耗。

7.1.2.3 矩阵运算的int8量化

代表浮点实数，代表量化后的定点整数。浮点数和整型之间的换算公式如下所示，其中 代表缩放系数，表示实数和整数之间的比例关系，分布代表浮点实数及定点整数最大最小值，代表实数中的0经过量化对应的整数值，为了在矩阵padding的时候保证浮点数值的0和定点整数的完全等价，保证定点和浮点之间的表征能够一致。

假设卷积的权重为，为，输入为，输出的激活值为。由于卷积本质上就是矩阵运算，因此表示成：

由此得到量化的公式：

这里面非整数部分就只有，因此接下来把这部分变成定点运算，对于由于 的结果通常会用int32的整数存储，因此通常也量化到int32，这里可以直接用来代表，由于都是对应8个bit的缩放比例，因此最多就放缩到16个bit ，用32bit来存放完全足够，直接记为0。因此公式调整为：

其中，通常为之间的实数，因此可表示为，其中是一个定点实数，这样可以通过的bit位移操作实现，这样整个过程就都在定点上计算了。

由于、、、都是可以事先计算的，因此、也可以事先计算好，实际推理的时候只需要计算、即可。

这里解释一下为什么可以用来代替，可以直接记为0：首先和只是充当和之间转换的桥梁，只要保证经过变换后得到，而这个经过和可以反变换得到即可。假设那么，一个区间的实数，完全可以通过和可以换算到区间的整数。如果对的范围限制到，和依然可以转换，但会把信息都压缩到更小的区间了。所以用来代替记为0。假设所有的，那么，而是用32bit存储的，本来，因此缩放系数砍掉了一半的信息量，会带来一定的精度损失，不过大部分情况下这点损失是可以忽略的。

## 实验环境要求

平台：Modelarts

操作系统：euler（aarch64）

软件环境：编程框架MindSpore1.7.1、异构计算架构CANN5.1，Python3.7.10

硬件环境：Ascend910

测试数据集：花卉数据集（雏菊、蒲公英、玫瑰、向日葵、郁金香）

## 实验详细设计与实现

### 7.3.1 实验步骤

对于一个简单的卷积网络，代表输入和输出，是网络中间的特征图，表示量化后的定点数，在后训练量化中，需要一些样本来统计的数值范围，在根据量化位数以及量化方法来计算和。



1. 首先输入部分样本图片进行正向传播，获取输入，输出及中间特征图的最大最小值。其中对于Relu、Maxpool算子来说，会沿用上一层输出的min和max，不需要额外统计。

2. 根据min和max以及量化的位数，计算和。

3. 在量化推理的时候，会把输入x化成定点整数，然后进行卷积计算，得到输出，这个结果依然是整型的，然后继续计算Relu的输出，对于FC也是矩阵运算，得到输出，根据计算出来的和推算回浮点整数，除了输入输出的量化与反量化操作，其他流程完全可以用定点运算来完成。

7.3.1.1 量化模块

首先实现基本的量化公式，具体代码如下

def calculate\_scale\_zero\_point(min\_val, max\_val, num\_bits=8):

qmin = 0.

qmax = 2. \*\* num\_bits - 1.

scale = float((max\_val - min\_val) / (qmax - qmin)) # S=(rmax-rmin)/(qmax-qmin)

zero\_point = round(qmax - max\_val / scale) # Z=round(qmax-rmax/scale)

if zero\_point < qmin:

zero\_point = qmin

elif zero\_point > qmax:

zero\_point = qmax

return scale, zero\_point

def quantize\_tensor(x, scale, zero\_point, num\_bits=8, signed=False):

# TODO 请根据公式实现张量的量化操作

return q\_x.astype(ms.float32)# 由于mindspore不支持int类型的运算，因此我们还是用float来表示整数

def dequantize\_tensor(q\_x, scale, zero\_point):

return scale \* (q\_x - zero\_point) # r=S(q-Z)

在量化过程中，需要先统计样本和中间层的最大最小值，同时也涉及到量化、反量化操作，因此将这些功能封装成一个QParam类。

class QParam(object):

def \_\_init\_\_(self, num\_bits=8):

self.num\_bits = num\_bits

self.scale = None

self.zero\_point = None

self.min = None

self.max = None

def update(self, tensor):

# 用来统计 min、max

if self.max is None or self.max < tensor.max():

self.max = tensor.max()

self.max = 0 if self.max < 0 else self.max

if self.min is None or self.min > tensor.min():

self.min = tensor.min()

self.min = 0 if self.min > 0 else self.min

self.scale, self.zero\_point = calculate\_scale\_zero\_point(self.min, self.max, self.num\_bits)

def quantize\_tensor(self, tensor):

return quantize\_tensor(tensor, self.scale, self.zero\_point, num\_bits=self.num\_bits)

def dequantize\_tensor(self, q\_x):

return dequantize\_tensor(q\_x, self.scale, self.zero\_point)

接着定义基本的量化基类

* \_\_init\_\_ 函数：指定量化的位数外，还需指定是否提供量化输入()及输出参数()。在前面也提到，不是每一个网络模块都需要统计输入的min、max，大部分中间层都是用上一层的来作为自己的的，另外有些中间层的激活函数也是直接用上一层的来作为自己的和。
* freeze函数：这个函数会在统计完min、max后发挥作用。正如上文所说的，公式(4)中有很多项是可以提前计算好的，freeze就是把这些项提前固定下来，同时也将网络的权重由浮点实数转化为定点整数。
* quantize\_inference函数：这个函数主要是量化inference的时候会使用。

class QModule(nn.Cell):

def \_\_init\_\_(self, qi=True, qo=True, num\_bits=8):

super(QModule, self).\_\_init\_\_()

if qi:

self.qi = QParam(num\_bits=num\_bits)

if qo:

self.qo = QParam(num\_bits=num\_bits)

def freeze(self):

pass

def quantize\_inference(self, x):

raise NotImplementedError('quantize\_inference should be implemented.')

7.3.1.2 量化卷积模块

量化卷积模块包括

* \_\_init\_\_函数：需要传入conv\_module模块，这个模块对应全精度的卷积层，另外参数则是用来统计weight的min、max以及对weight进行量化用的。
* freeze函数：这个函数主要就是计算公式中的 ，其中应该由移位来实现定点化加速，为了实现方便，在此用原始的数学操作进行代替。
* construct函数：这个函数和正常的construct一样，也是在float上进行的，只不过需要统计输入输出以及weight的min、max而已。其中这里需要对weight量化到int8然后又反量化回float，这里其实就是所谓的伪量化节点，因为我们在实际量化inference的时候会把weight量化到int8，这个过程本身是有精度损失的 (来自四舍五入的round带来的截断误差)，所以在统计min、max的时候，需要把这个过程带来的误差也模拟进去。
* quantize\_inference函数：这个函数在实际inference的时候会被调用。注意，这个函数里面的卷积操作是在int上进行的，这是量化推理加速的关键「当然，由于mindspore的限制，我们仍然是在float上计算，只不过数值都是整数。这也可以看出量化推理是跟底层实现紧密结合的技术」。

class QConv2d(QModule):

def \_\_init\_\_(self, conv\_module, qi=True, qo=True, num\_bits=8):

super(QConv2d, self).\_\_init\_\_(qi=qi, qo=qo, num\_bits=num\_bits)

self.num\_bits = num\_bits

self.conv\_module = conv\_module

self.qw = QParam(num\_bits=num\_bits)

self.M = None

def freeze(self, qi=None, qo=None):

if hasattr(self, 'qi') and qi is not None:

raise ValueError('qi has been provided in init function.')

if not hasattr(self, 'qi') and qi is None:

raise ValueError('qi is not existed, should be provided.')

if hasattr(self, 'qo') and qo is not None:

raise ValueError('qo has been provided in init function.')

if not hasattr(self, 'qo') and qo is None:

raise ValueError('qo is not existed, should be provided.')

if qi is not None:

self.qi = qi

if qo is not None:

self.qo = qo

# TODO 请实现卷积模块权重参数量化

self.M =

self.conv\_module.weight =

self.conv\_module.bias =

def construct(self, x):

if hasattr(self, 'qi'):

self.qi.update(x)

x = self.qi.quantize\_tensor(x)

x = self.qi.dequantize\_tensor(x)

self.qw.update(self.conv\_module.weight)

self.conv\_module.weight = self.qw.quantize\_tensor(self.conv\_module.weight)

self.conv\_module.weight = self.qw.dequantize\_tensor(self.conv\_module.weight)

x = ops.conv2d(x, self.conv\_module.weight, stride=self.conv\_module.stride, pad\_mode=self.conv\_module.pad\_mode)

if self.conv\_module.bias is not None:

x = ops.bias\_add(x, self.conv\_module.bias)

if hasattr(self, 'qo'):

self.qo.update(x)

x = self.qo.quantize\_tensor(x)

x = self.qo.dequantize\_tensor(x)

return x

def quantize\_inference(self, x):

# TODO 请实现卷积模块量化推理模块

return x

7.5.1.3 量化全连接层模块

与量化卷积模块功能相似，这里不再叙述。

class QDense(QModule):

def \_\_init\_\_(self, fc\_module, qi=True, qo=True, num\_bits=8):

super(QDense, self).\_\_init\_\_(qi=qi, qo=qo, num\_bits=num\_bits)

self.num\_bits = num\_bits

self.fc\_module = fc\_module

self.qw = QParam(num\_bits=num\_bits)

self.M = ms.Tensor([])

def freeze(self, qi=None, qo=None):

if hasattr(self, 'qi') and qi is not None:

raise ValueError('qi has been provided in init function.')

if not hasattr(self, 'qi') and qi is None:

raise ValueError('qi is not existed, should be provided.')

if hasattr(self, 'qo') and qo is not None:

raise ValueError('qo has been provided in init function.')

if not hasattr(self, 'qo') and qo is None:

raise ValueError('qo is not existed, should be provided.')

if qi is not None:

self.qi = qi

if qo is not None:

self.qo = qo

# TODO 请实现全连接模块权重参数量化

self.M =

self.fc\_module.weight =

self.fc\_module.bias =

def construct(self, x):

if hasattr(self, 'qi'):

self.qi.update(x)

x = self.qi.quantize\_tensor(x)

x = self.qi.dequantize\_tensor(x)

self.qw.update(self.fc\_module.weight)

self.fc\_module.weight = self.qw.quantize\_tensor(self.fc\_module.weight)

self.fc\_module.weight = self.qw.dequantize\_tensor(self.fc\_module.weight)

x = ops.matmul(x, self.fc\_module.weight.T)

x = ops.bias\_add(x, self.fc\_module.bias)

if hasattr(self, 'qo'):

self.qo.update(x)

x = self.qo.quantize\_tensor(x)

x = self.qo.dequantize\_tensor(x)

return x

def quantize\_inference(self, x):

# TODO 请实现全连接模块量化推理

return x

7.5.1.4 量化ReLU模块

大体内容与量化卷积模块相似，其中需要注意，在 quantize\_inference 函数中，量化零点非真实的0，需要特别注意。

class QReLU(QModule):

def \_\_init\_\_(self, qi=False, num\_bits=None):

super(QReLU, self).\_\_init\_\_(qi=qi, num\_bits=num\_bits)

def freeze(self, qi=None):

if hasattr(self, 'qi') and qi is not None:

raise ValueError('qi has been provided in init function.')

if not hasattr(self, 'qi') and qi is None:

raise ValueError('qi is not existed, should be provided.')

if qi is not None:

self.qi = qi

def construct(self, x):

if hasattr(self, 'qi'):

self.qi.update(x)

x = self.qi.quantize\_tensor(x)

x = self.qi.dequantize\_tensor(x)

x = ops.relu(x)

return x

def quantize\_inference(self, x):

x[x < self.qi.zero\_point] = self.qi.zero\_point

return x

7.5.1.5 量化最大池化模块

大体内容与量化卷积模块相似，在量化推理时，因为最大池化原理就是取区域最大值作为输出，故直接进行算子运算即可。

class QMaxPooling2d(QModule):

def \_\_init\_\_(self, max\_pool\_module, qi=False, num\_bits=None):

super(QMaxPooling2d, self).\_\_init\_\_(qi=qi, num\_bits=num\_bits)

self.max\_pool\_module = max\_pool\_module

def freeze(self, qi=None):

if hasattr(self, 'qi') and qi is not None:

raise ValueError('qi has been provided in init function.')

if not hasattr(self, 'qi') and qi is None:

raise ValueError('qi is not existed, should be provided.')

if qi is not None:

self.qi = qi

def construct(self, x):

if hasattr(self, 'qi'):

self.qi.update(x)

x = self.qi.quantize\_tensor(x)

x = self.qi.dequantize\_tensor(x)

x = self.max\_pool\_module(x)

return x

def quantize\_inference(self, x):

return self.max\_pool\_module(x)

7.5.1.6 量化VGG网络

量化卷积模块包括

* \_\_init\_\_ 函数：基本的算子定义
* construct 函数：网络正向传播模块
* quantize 函数：量化网络模块
* quantize\_forward 函数：量化正向传播模块
* freeze 函数：量化参数冻结模块
* quantize\_inference 函数：量化推理模块

import mindspore.nn as nn

from quant\_module import QConv2d, QMaxPooling2d, QDense, QReLU

class Vgg(nn.Cell):

def \_\_init\_\_(self, num\_classes=4):

super(Vgg, self).\_\_init\_\_()

self.layer1\_conv1 = nn.Conv2d(in\_channels=3, out\_channels=64, kernel\_size=3, has\_bias=True)

self.layer1\_relu1 = nn.ReLU()

self.layer1\_conv2 = nn.Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3, has\_bias=True)

self.layer1\_relu2 = nn.ReLU()

self.layer1\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer2\_conv1 = nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, has\_bias=True)

self.layer2\_relu1 = nn.ReLU()

self.layer2\_conv2 = nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, has\_bias=True)

self.layer2\_relu2 = nn.ReLU()

self.layer2\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer3\_conv1 = nn.Conv2d(in\_channels=128, out\_channels=256, kernel\_size=3, has\_bias=True)

self.layer3\_relu1 = nn.ReLU()

self.layer3\_conv2 = nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, has\_bias=True)

self.layer3\_relu2 = nn.ReLU()

self.layer3\_conv3 = nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, has\_bias=True)

self.layer3\_relu3 = nn.ReLU()

self.layer3\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer4\_conv1 = nn.Conv2d(in\_channels=256, out\_channels=512, kernel\_size=3, has\_bias=True)

self.layer4\_relu1 = nn.ReLU()

self.layer4\_conv2 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, has\_bias=True)

self.layer4\_relu2 = nn.ReLU()

self.layer4\_conv3 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, has\_bias=True)

self.layer4\_relu3 = nn.ReLU()

self.layer4\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer5\_conv1 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, has\_bias=True)

self.layer5\_relu1 = nn.ReLU()

self.layer5\_conv2 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, has\_bias=True)

self.layer5\_relu2 = nn.ReLU()

self.layer5\_conv3 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, has\_bias=True)

self.layer5\_relu3 = nn.ReLU()

self.layer5\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

self.fullyconnect1 = nn.Dense(512 \* 7 \* 7, 4096)

self.relu\_1 = nn.ReLU()

self.fullyconnect2 = nn.Dense(4096, 4096)

self.relu\_2 = nn.ReLU()

self.fullyconnect3 = nn.Dense(4096, num\_classes)

def construct(self, x):

x = self.layer1\_conv1(x)

x = self.layer1\_relu1(x)

x = self.layer1\_conv2(x)

x = self.layer1\_relu2(x)

x = self.layer1\_maxpool(x)

x = self.layer2\_conv1(x)

x = self.layer2\_relu1(x)

x = self.layer2\_conv2(x)

x = self.layer2\_relu2(x)

x = self.layer2\_maxpool(x)

x = self.layer3\_conv1(x)

x = self.layer3\_relu1(x)

x = self.layer3\_conv2(x)

x = self.layer3\_relu2(x)

x = self.layer3\_conv3(x)

x = self.layer3\_relu3(x)

x = self.layer3\_maxpool(x)

x = self.layer4\_conv1(x)

x = self.layer4\_relu1(x)

x = self.layer4\_conv2(x)

x = self.layer4\_relu2(x)

x = self.layer4\_conv3(x)

x = self.layer4\_relu3(x)

x = self.layer4\_maxpool(x)

x = self.layer5\_conv1(x)

x = self.layer5\_relu1(x)

x = self.layer5\_conv2(x)

x = self.layer5\_relu2(x)

x = self.layer5\_conv3(x)

x = self.layer5\_relu3(x)

x = self.layer5\_maxpool(x)

x = self.flatten(x)

x = self.fullyconnect1(x)

x = self.relu\_1(x)

x = self.fullyconnect2(x)

x = self.relu\_2(x)

x = self.fullyconnect3(x)

return x

def quantize(self, num\_bits=8):

# 第一个卷积模块需要获取量化输入，其余模块会复用之前的量化输出

self.qlayer1\_conv1 = QConv2d(self.layer1\_conv1, qi=True, qo=True, num\_bits=num\_bits)

self.qlayer1\_relu1 = QReLU()

self.qlayer1\_conv2 = QConv2d(self.layer1\_conv2, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer1\_relu2 = QReLU()

self.qlayer1\_maxpool2d = QMaxPooling2d(self.layer1\_maxpool)

self.qlayer2\_conv1 = QConv2d(self.layer2\_conv1, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer2\_relu1 = QReLU()

self.qlayer2\_conv2 = QConv2d(self.layer2\_conv2, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer2\_relu2 = QReLU()

self.qlayer2\_maxpool2d = QMaxPooling2d(self.layer2\_maxpool)

self.qlayer3\_conv1 = QConv2d(self.layer3\_conv1, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer3\_relu1 = QReLU()

self.qlayer3\_conv2 = QConv2d(self.layer3\_conv2, qi=False, qo=True,num\_bits=num\_bits)

self.qlayer3\_relu2 = QReLU()

self.qlayer3\_conv3 = QConv2d(self.layer3\_conv3, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer3\_relu3 = QReLU()

self.qlayer3\_maxpool2d = QMaxPooling2d(self.layer3\_maxpool)

self.qlayer4\_conv1 = QConv2d(self.layer4\_conv1, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer4\_relu1 = QReLU()

self.qlayer4\_conv2 = QConv2d(self.layer4\_conv2, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer4\_relu2 = QReLU()

self.qlayer4\_conv3 = QConv2d(self.layer4\_conv3, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer4\_relu3 = QReLU()

self.qlayer4\_maxpool2d = QMaxPooling2d(self.layer4\_maxpool)

self.qlayer5\_conv1 = QConv2d(self.layer5\_conv1, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer5\_relu1 = QReLU()

self.qlayer5\_conv2 = QConv2d(self.layer5\_conv2, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer5\_relu2 = QReLU()

self.qlayer5\_conv3 = QConv2d(self.layer5\_conv3, qi=False, qo=True, num\_bits=num\_bits)

self.qlayer5\_relu3 = QReLU()

self.qlayer5\_maxpool2d = QMaxPooling2d(self.layer5\_maxpool)

self.qfc1 = QDense(self.fullyconnect1, qi=False, qo=True, num\_bits=num\_bits)

self.qfc1\_relu = QReLU()

self.qfc2 = QDense(self.fullyconnect2, qi=False, qo=True, num\_bits=num\_bits)

self.qfc2\_relu = QReLU()

self.qfc3 = QDense(self.fullyconnect3, qi=False, qo=True, num\_bits=num\_bits)

def quantize\_forward(self, x):

x = self.qlayer1\_conv1(x)

x = self.qlayer1\_relu1(x)

x = self.qlayer1\_conv2(x)

x = self.qlayer1\_relu2(x)

x = self.qlayer1\_maxpool2d(x)

x = self.qlayer2\_conv1(x)

x = self.qlayer2\_relu1(x)

x = self.qlayer2\_conv2(x)

x = self.qlayer2\_relu2(x)

x = self.qlayer2\_maxpool2d(x)

x = self.qlayer3\_conv1(x)

x = self.qlayer3\_relu1(x)

x = self.qlayer3\_conv2(x)

x = self.qlayer3\_relu2(x)

x = self.qlayer3\_conv3(x)

x = self.qlayer3\_relu3(x)

x = self.qlayer3\_maxpool2d(x)

x = self.qlayer4\_conv1(x)

x = self.qlayer4\_relu1(x)

x = self.qlayer4\_conv2(x)

x = self.qlayer4\_relu2(x)

x = self.qlayer4\_conv3(x)

x = self.qlayer4\_relu3(x)

x = self.qlayer4\_maxpool2d(x)

x = self.qlayer5\_conv1(x)

x = self.qlayer5\_relu1(x)

x = self.qlayer5\_conv2(x)

x = self.qlayer5\_relu2(x)

x = self.qlayer5\_conv3(x)

x = self.qlayer5\_relu3(x)

x = self.qlayer5\_maxpool2d(x)

x = self.flatten(x)

x = self.qfc1(x)

x = self.qfc1\_relu(x)

x = self.qfc2(x)

x = self.qfc2\_relu(x)

x = self.qfc3(x)

return x

def freeze(self):

# 冻结网络参数时，除第一个卷积模块不需要指定量化输入，其余模块都需要指定量化输入

self.qlayer1\_conv1.freeze()

self.qlayer1\_relu1.freeze(qi=self.qlayer1\_conv1.qo)

self.qlayer1\_conv2.freeze(qi=self.qlayer1\_conv1.qo)

self.qlayer1\_relu2.freeze(qi=self.qlayer1\_conv2.qo)

self.qlayer1\_maxpool2d.freeze(qi=self.qlayer1\_conv2.qo)

self.qlayer2\_conv1.freeze(qi=self.qlayer1\_conv2.qo)

self.qlayer2\_relu1.freeze(qi=self.qlayer2\_conv1.qo)

self.qlayer2\_conv2.freeze(qi=self.qlayer2\_conv1.qo)

self.qlayer2\_relu2.freeze(qi=self.qlayer2\_conv2.qo)

self.qlayer2\_maxpool2d.freeze(qi=self.qlayer2\_conv2.qo)

self.qlayer3\_conv1.freeze(qi=self.qlayer2\_conv2.qo)

self.qlayer3\_relu1.freeze(qi=self.qlayer3\_conv1.qo)

self.qlayer3\_conv2.freeze(qi=self.qlayer3\_conv1.qo)

self.qlayer3\_relu2.freeze(qi=self.qlayer3\_conv2.qo)

self.qlayer3\_conv3.freeze(qi=self.qlayer3\_conv2.qo)

self.qlayer3\_relu3.freeze(qi=self.qlayer3\_conv3.qo)

self.qlayer3\_maxpool2d.freeze(qi=self.qlayer3\_conv3.qo)

self.qlayer4\_conv1.freeze(qi=self.qlayer3\_conv3.qo)

self.qlayer4\_relu1.freeze(qi=self.qlayer4\_conv1.qo)

self.qlayer4\_conv2.freeze(qi=self.qlayer4\_conv1.qo)

self.qlayer4\_relu2.freeze(qi=self.qlayer4\_conv2.qo)

self.qlayer4\_conv3.freeze(qi=self.qlayer4\_conv2.qo)

self.qlayer4\_relu3.freeze(qi=self.qlayer4\_conv3.qo)

self.qlayer4\_maxpool2d.freeze(qi=self.qlayer4\_conv3.qo)

self.qlayer5\_conv1.freeze(qi=self.qlayer4\_conv3.qo)

self.qlayer5\_relu1.freeze(qi=self.qlayer5\_conv1.qo)

self.qlayer5\_conv2.freeze(qi=self.qlayer5\_conv1.qo)

self.qlayer5\_relu2.freeze(qi=self.qlayer5\_conv2.qo)

self.qlayer5\_conv3.freeze(qi=self.qlayer5\_conv2.qo)

self.qlayer5\_relu3.freeze(qi=self.qlayer5\_conv3.qo)

self.qlayer5\_maxpool2d.freeze(qi=self.qlayer5\_conv3.qo)

self.qfc1.freeze(qi=self.qlayer5\_conv3.qo)

self.qfc1\_relu.freeze(qi=self.qfc1.qo)

self.qfc2.freeze(qi=self.qfc1.qo)

self.qfc2\_relu.freeze(qi=self.qfc2.qo)

self.qfc3.freeze(qi=self.qfc2.qo)

def quantize\_inference(self, x):

# 对输入x进行量化

qx = self.qlayer1\_conv1.qi.quantize\_tensor(x)

qx = self.qlayer1\_conv1.quantize\_inference(qx)

qx = self.qlayer1\_relu1.quantize\_inference(qx)

qx = self.qlayer1\_conv2.quantize\_inference(qx)

qx = self.qlayer1\_relu2.quantize\_inference(qx)

qx = self.qlayer1\_maxpool2d.quantize\_inference(qx)

qx = self.qlayer2\_conv1.quantize\_inference(qx)

qx = self.qlayer2\_relu1.quantize\_inference(qx)

qx = self.qlayer2\_conv2.quantize\_inference(qx)

qx = self.qlayer2\_relu2.quantize\_inference(qx)

qx = self.qlayer2\_maxpool2d.quantize\_inference(qx)

qx = self.qlayer3\_conv1.quantize\_inference(qx)

qx = self.qlayer3\_relu1.quantize\_inference(qx)

qx = self.qlayer3\_conv2.quantize\_inference(qx)

qx = self.qlayer3\_relu2.quantize\_inference(qx)

qx = self.qlayer3\_conv3.quantize\_inference(qx)

qx = self.qlayer3\_relu3.quantize\_inference(qx)

qx = self.qlayer3\_maxpool2d.quantize\_inference(qx)

qx = self.qlayer4\_conv1.quantize\_inference(qx)

qx = self.qlayer4\_relu1.quantize\_inference(qx)

qx = self.qlayer4\_conv2.quantize\_inference(qx)

qx = self.qlayer4\_relu2.quantize\_inference(qx)

qx = self.qlayer4\_conv3.quantize\_inference(qx)

qx = self.qlayer4\_relu3.quantize\_inference(qx)

qx = self.qlayer4\_maxpool2d.quantize\_inference(qx)

qx = self.qlayer5\_conv1.quantize\_inference(qx)

qx = self.qlayer5\_relu1.quantize\_inference(qx)

qx = self.qlayer5\_conv2.quantize\_inference(qx)

qx = self.qlayer5\_relu2.quantize\_inference(qx)

qx = self.qlayer5\_conv3.quantize\_inference(qx)

qx = self.qlayer5\_relu3.quantize\_inference(qx)

qx = self.qlayer5\_maxpool2d.quantize\_inference(qx)

qx = self.flatten(qx)

qx = self.qfc1.quantize\_inference(qx)

qx = self.qfc1\_relu.quantize\_inference(qx)

qx = self.qfc2.quantize\_inference(qx)

qx = self.qfc2\_relu.quantize\_inference(qx)

qx = self.qfc3.quantize\_inference(qx)

# 对输出qx进行反量化

out = self.qfc3.qo.dequantize\_tensor(qx)

return out

### 7.3.2 实验流程

第一步：初始化VGG网络并加载权重系数

第二步：构建对应推理数据

第三步：首先进行正常的网络推理，获取模型输出

第四步：构建量化模型，此实验为int8量化

第五步：进行量化推理，这里涉及到对中间特征图统计最大最小值

第六步：对网络量化参数进行固定

第七步：进行量化推理

import numpy as np

import cv2

import mindspore as ms

from mindspore import ops

from mindspore import load\_checkpoint, load\_param\_into\_net

from mindspore import context

from vgg import Vgg

context.set\_context(mode=context.PYNATIVE\_MODE, device\_target='CPU')

np.set\_printoptions(suppress=True)

def resize\_image(image, target\_size):

h, w = image.shape[:2]

th, tw = target\_size

# 获取等比缩放后的尺寸

scale = min(th / h, tw / w)

oh, ow = round(h \* scale), round(w \* scale)

# 缩放图片，opencv缩放传入尺寸为（宽，高），这里采用线性差值算法

image = cv2.resize(image, (ow, oh), interpolation=cv2.INTER\_LINEAR).astype(np.uint8)

# 将剩余部分进行填充

new\_image = np.ones((th, tw, 3), dtype=np.uint8) \* 114

new\_image[:oh, :ow, :] = image

return new\_image

def process\_image(img\_path):

# 读取图片，opencv读图后格式是BGR格式，需要转为RGB格式

image = cv2.imread(img\_path, cv2.IMREAD\_COLOR)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# 将图片等比resize至(224x224)

image = resize\_image(image, (224, 224))

image = np.array(image, dtype=np.float32)

# 将图片标准化

image -= [125.307, 122.961, 113.8575]

image /= [51.5865, 50.847, 51.255]

# (h,w,c) -> (c,h,w) -> (1,c,h,w)

image = image.transpose((2, 0, 1))[None]

return image

def direct\_quantize(model, dataset):

print('\*'\*50)

print('Start quantize')

for img\_path, label in dataset:

print("Start inference: {}".format(img\_path))

ndarray = process\_image(img\_path)

tensor = ms.Tensor(ndarray, ms.float32)

net\_out = model.quantize\_forward(tensor)

prob = ops.Softmax()(net\_out)

print('Predict probability: {}'.format(np.around(prob.asnumpy(), 4)))

predict\_cls = (ops.Argmax()(prob)).asnumpy().item()

print('Inference result: {}\n'.format(predict\_cls == label))

def full\_inference(model, dataset):

print('\*' \* 50)

print('Start full inference')

for img\_path, label in dataset:

print("Start inference: {}".format(img\_path))

ndarray = process\_image(img\_path)

tensor = ms.Tensor(ndarray, ms.float32)

net\_out = model(tensor)

prob = ops.Softmax()(net\_out)

print('Predict probability: {}'.format(np.around(prob.asnumpy(), 4)))

predict\_cls = (ops.Argmax()(prob)).asnumpy().item()

print('Inference result: {}\n'.format(predict\_cls == label))

def quantize\_inference(model, dataset):

print('\*' \* 50)

print('Start quantize inference')

for img\_path, label in dataset:

print("Start inference: {}".format(img\_path))

ndarray = process\_image(img\_path)

tensor = ms.Tensor(ndarray, ms.float32)

net\_out = model.quantize\_inference(tensor)

prob = ops.Softmax()(net\_out)

print('Predict probability: {}'.format(np.around(prob.asnumpy(), 4)))

predict\_cls = (ops.Argmax()(prob)).asnumpy().item()

print('Inference result: {}\n'.format(predict\_cls == label))

if \_\_name\_\_ == '\_\_main\_\_':

# 初始化VGG网络并加载权重系数

net = Vgg(num\_classes=4)

load\_param\_into\_net(net, load\_checkpoint('vgg.ckpt'), strict\_load=True)

net.set\_train(False)

# 构建对应推理数据

dataset = [('./data/daisy\_demo.jpg', 0),

('./data/roses\_demo.jpg', 1),

('./data/sunflowers\_demo.jpg', 2),

('./data/tulips\_demo.jpg', 3)]

# 首先进行正常的网络推理，获取模型输出

full\_inference(net, dataset)

# 构建量化模型，此实验为int8量化

net.quantize(num\_bits=8)

# 进行量化推理，这里涉及到对中间特征图统计最大最小值

direct\_quantize(net, dataset)

# 对网络量化参数进行固定

net.freeze()

# 进行量化推理

quantize\_inference(net, dataset)

### 7.3.3 实验运行

代码目录介绍：

EXP

|- data

| |- daisy\_demo.jpg # 测试图片

| |- roses\_demo.jpg # 测试图片

| |- sunflowers\_demo.jpg # 测试图片

| |- daisy\_demo.jpg # 测试图片

|- quant\_inference.py # 量化推理主函数

|- quant\_module.py # 量化模块

|- vgg.ckpt # VGG网络权重

|- vgg.py # VGG网络模块

1. 实现代码

vim quant\_module.py

vim vgg.py

1. 运行实验

python3.7 quant\_inference.py

## 实验总结

本实验的目的是了解神经网络量化操作，能够独立实现int8量化操作，构建量化VGG16神经网络，并基于MindSpore框架实现量化推理，能够独立编写量化操作代码。同时同学们可以拓展思考，在推理的时候，为了实现网络加速优化，通常将Convolution算子、BatchNormalization算子、Relu算子进行算子融合。若不进行融合，请思考如何实现BatchNormalization的量化模块。

## 实验任务与参考解答任务

### 实验任务

本实验需要同学们能够独立实现int8量化操作，构建量化VGG16神经网络，并基于MindSpore框架实现量化推理，能够独立编写量化操作代码。

### 参考答案

量化模块：

def calculate\_scale\_zero\_point(min\_val, max\_val, num\_bits=8):

qmin = 0.

qmax = 2. \*\* num\_bits - 1.

scale = float((max\_val - min\_val) / (qmax - qmin)) # S=(rmax-rmin)/(qmax-qmin)

zero\_point = round(qmax - max\_val / scale) # Z=round(qmax-rmax/scale)

if zero\_point < qmin:

zero\_point = qmin

elif zero\_point > qmax:

zero\_point = qmax

return scale, zero\_point

def quantize\_tensor(x, scale, zero\_point, num\_bits=8, signed=False):

# TODO 请根据公式实现张量的量化操作

if signed:

qmin = - 2. \*\* (num\_bits - 1)

qmax = 2. \*\* (num\_bits - 1) - 1

else:

qmin = 0. qmax = 2. \*\* num\_bits - 1.

q\_x = zero\_point + x / scale

q\_x = (q\_x.clip(qmin, qmax)).round() # q=round(r/S+Z)

return q\_x.astype(ms.float32)# 由于mindspore不支持int类型的运算，因此我们还是用float来表示整数

def dequantize\_tensor(q\_x, scale, zero\_point):

return scale \* (q\_x - zero\_point) # r=S(q-Z)

量化卷积模块：

class QConv2d(QModule):

def \_\_init\_\_(self, conv\_module, qi=True, qo=True, num\_bits=8):

super(QConv2d, self).\_\_init\_\_(qi=qi, qo=qo, num\_bits=num\_bits)

self.num\_bits = num\_bits

self.conv\_module = conv\_module

self.qw = QParam(num\_bits=num\_bits)

self.M = None

def freeze(self, qi=None, qo=None):

if hasattr(self, 'qi') and qi is not None:

raise ValueError('qi has been provided in init function.')

if not hasattr(self, 'qi') and qi is None:

raise ValueError('qi is not existed, should be provided.')

if hasattr(self, 'qo') and qo is not None:

raise ValueError('qo has been provided in init function.')

if not hasattr(self, 'qo') and qo is None:

raise ValueError('qo is not existed, should be provided.')

if qi is not None:

self.qi = qi

if qo is not None:

self.qo = qo

# TODO 请实现卷积模块权重参数量化

self.M = self.qw.scale \* self.qi.scale / self.qo.scale

self.conv\_module.weight = self.qw.quantize\_tensor(self.conv\_module.weight)

self.conv\_module.weight = self.conv\_module.weight - self.qw.zero\_point

self.conv\_module.bias = quantize\_tensor(self.conv\_module.bias,

scale=self.qi.scale \* self.qw.scale,

zero\_point=0,

num\_bits=32, signed=True)

def construct(self, x):

if hasattr(self, 'qi'):

self.qi.update(x)

x = self.qi.quantize\_tensor(x)

x = self.qi.dequantize\_tensor(x)

self.qw.update(self.conv\_module.weight)

self.conv\_module.weight = self.qw.quantize\_tensor(self.conv\_module.weight)

self.conv\_module.weight = self.qw.dequantize\_tensor(self.conv\_module.weight)

x = ops.conv2d(x, self.conv\_module.weight, stride=self.conv\_module.stride, pad\_mode=self.conv\_module.pad\_mode)

if self.conv\_module.bias is not None:

x = ops.bias\_add(x, self.conv\_module.bias)

if hasattr(self, 'qo'):

self.qo.update(x)

x = self.qo.quantize\_tensor(x)

x = self.qo.dequantize\_tensor(x)

return x

def quantize\_inference(self, x):

# TODO 请实现卷积模块量化推理模块

x = x - self.qi.zero\_point

x = self.conv\_module(x)

x = self.M \* x

x = x.round()

x = x + self.qo.zero\_point

x = x.clip(0., 2. \*\* self.num\_bits - 1.).round()

return x

量化全连接模块：

class QDense(QModule):

def \_\_init\_\_(self, fc\_module, qi=True, qo=True, num\_bits=8):

super(QDense, self).\_\_init\_\_(qi=qi, qo=qo, num\_bits=num\_bits)

self.num\_bits = num\_bits

self.fc\_module = fc\_module

self.qw = QParam(num\_bits=num\_bits)

self.M = ms.Tensor([])

def freeze(self, qi=None, qo=None):

if hasattr(self, 'qi') and qi is not None:

raise ValueError('qi has been provided in init function.')

if not hasattr(self, 'qi') and qi is None:

raise ValueError('qi is not existed, should be provided.')

if hasattr(self, 'qo') and qo is not None:

raise ValueError('qo has been provided in init function.')

if not hasattr(self, 'qo') and qo is None:

raise ValueError('qo is not existed, should be provided.')

if qi is not None:

self.qi = qi

if qo is not None:

self.qo = qo

# TODO 请实现全连接模块权重参数量化

self.M = self.qw.scale \* self.qi.scale / self.qo.scale

self.fc\_module.weight = self.qw.quantize\_tensor(self.fc\_module.weight)

self.fc\_module.weight = self.fc\_module.weight.data - self.qw.zero\_point

self.fc\_module.bias = quantize\_tensor(self.fc\_module.bias,

scale=self.qi.scale \* self.qw.scale,

zero\_point=0,

num\_bits=32, signed=True)

def construct(self, x):

if hasattr(self, 'qi'):

self.qi.update(x)

x = self.qi.quantize\_tensor(x)

x = self.qi.dequantize\_tensor(x)

self.qw.update(self.fc\_module.weight)

self.fc\_module.weight = self.qw.quantize\_tensor(self.fc\_module.weight)

self.fc\_module.weight = self.qw.dequantize\_tensor(self.fc\_module.weight)

x = ops.matmul(x, self.fc\_module.weight.T)

x = ops.bias\_add(x, self.fc\_module.bias)

if hasattr(self, 'qo'):

self.qo.update(x)

x = self.qo.quantize\_tensor(x)

x = self.qo.dequantize\_tensor(x)

return x

def quantize\_inference(self, x):

# TODO 请实现全连接模块量化推理

x = x - self.qi.zero\_point

x = self.fc\_module(x)

x = self.M \* x

x = x.round()

x = x + self.qo.zero\_point

x = x.clip(0., 2. \*\* self.num\_bits - 1.).round()

return x