CANN
6.3.RC2

PyTorch 在线推理使用指南

文档版本 01

发布日期 2023-07-25

华为技术有限公司

商标声明


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： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 简介

在线推理是在AI框架内执行推理的场景，例如在PyTorch框架上，加载模型后，通过**model.eval()**将模型切换为在线推理模式。相比于离线推理场景，使用在线推理可以方便将原来基于PyTorch框架做推理的应用快速迁移到昇腾AI处理器，适用于数据中心推理场景。

支持的芯片型号

- 昇腾910 AI处理器
- 昇腾910B AI处理器

2 使用流程

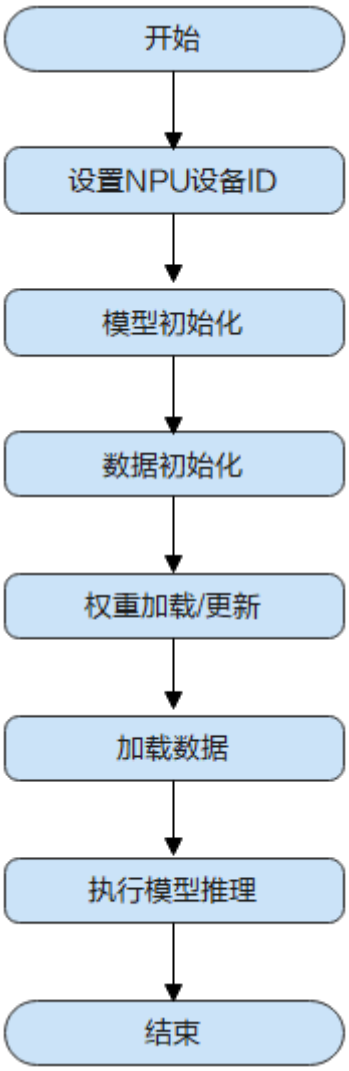
2.1 前提条件

已完成PyTorch框架及混合精度模块的安装，请用户参考《**CANN 软件安装指南**》中的“安装须知”章节选择安装开发环境（Ascend-cann-toolkit）或运行环境nnae软件（Ascend-cann-nnae），并参考其中的“在昇腾设备上安装>安装深度学习框架>安装PyTorch”章节安装PyTorch框架。

2.2 在线推理流程

在线推理流程如**图2-1**所示。

图 2-1 在线推理流程图



2.3 配置环境变量

PyTorch在线推理所依赖的环境变量和配置如下：

- 请依据实际在下列场景中选择其一，进行在线推理依赖包安装路径的环境变量设置。具体如下（以HwHiAiUser用户安装，安装路径为默认路径为例）：
  - 场景一：昇腾设备安装部署开发套件包Ascend-cann-toolkit（此时开发环境可进行推理任务）。
    ```
    . /home/HwHiAiUser/Ascend/ascend-toolkit/set_env.sh
    ```
  - 场景二：昇腾设备安装部署软件包Ascend-cann-nnae。此时需要参考《**CANN 软件安装指南**》中的“安装运行环境（nnae软件，在物理机安装）”章节安装Ascend-cann-nnae。
    ```
    . /home/HwHiAiUser/Ascend/nnae/set_env.sh
    ```
- 若运行环境中存在多个Python3版本时，需要在环境变量中配置指定Python版本的安装路径。以Python3.7.5为例：
  ```
  export PATH=/usr/local/python3.7.5/bin:$PATH
  export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
  ```


- 指定芯片的逻辑ID。
  ```
  export ASCEND_DEVICE_ID=0
  ```
- 输出日志信息，可根据实际修改。
  ```
  export ASCEND_SLOG_PRINT_TO_STDOUT=1
  export ASCEND_GLOBAL_LOG_LEVEL=0
  ```

表 2-1 环境变量说明表

配置项	说明	必选/可选
LD_LIBRARY_PATH	动态库的查找路径，参考上述举例配置。 **说明** 若系统环境安装了gcc7.3.0（例如CentOS7.6、Debian和BClinux系统），需要配置gcc相关环境变量。 `export LD_LIBRARY_PATH=${install_path}/lib64:${LD_LIBRARY_PATH}` 其中${install_path}为gcc7.3.0安装路径。	必选
PATH	可执行程序的查找路径，参考上述举例配置。	必选
ASCEND_DEVICE_ID	指定芯片的逻辑ID。取值范围[0,N-1]，默认为0。其中N为当前物理机/虚拟机/容器内的设备总数。	可选
ASCEND_SLOG_PRINT_TO_STDOUT	是否开启日志打屏。 ● 0或不配置：关闭日志打屏 ● 1：开启日志打屏	可选
ASCEND_GLOBAL_LOG_LEVEL	设置日志的全局日志级别。 ● 0：对应DEBUG级别。 ● 1：对应INFO级别。 ● 2：对应WARNING级别。 ● 3：对应ERROR级别。 ● 4：对应NULL级别，不输出日志。 ● 其他值为非法值。	可选

更多日志信息，请参见《**日志参考**》。

2.4 使能混合精度

概述

混合精度训练是在训练时混合使用单精度（float32）与半精度(float16)数据类型，将两者结合在一起，并使用相同的超参数实现了与float32几乎相同的精度。若用户使用昇腾910 AI处理器，则在迁移完成、训练开始之前，由于其架构特性限制，用户需要开启混合精度。若用户使用昇腾910B AI处理器，则可以选择是否开启混合精度。使用float16代替float32有如下好处：

- 对于中间变量的内存占用更少，节省内存的使用。
- 因内存使用会减少，所以数据传出的时间也会相应减少。
- float16的计算单元可以提供更快的计算性能。

但是，混合精度训练受限于float16表达的精度范围，单纯将float32转换成float16会影响训练收敛情况。为了保证部分计算使用float16来进行加速的同时能保证训练收敛，这里推荐采用混合精度模块APEX来达到以上效果。混合精度模块APEX是一个集优化性能、精度收敛于一身的综合优化库。用户也可以选择使用PyTorch1.8.1及以上版本框架内置的AMP功能模块来使能混合精度。

更多混合精度原理介绍可参考《**PyTorch模型迁移和训练指南**》中“模型迁移与训练>自动混合精度（AMP）>概述”章节。

安装模块

混合精度模块的安装请参考《**CANN 软件安装指南**》中“安装开发环境>在昇腾设备上安装>安装深度学习框架>安装PyTorch”章节。

特性支持

混合精度模块功能和优化描述如**表2-2**所示。

表 2-2 混合精度模块功能

功能	开启方式举例	描述
O1配置模式	model, optimizer = amp.initialize(model, optimizer, **opt_level="O1"**)	• 白名单：使用Cube加速的算子，总是使用float16计算，例如Conv2d、Matmul。 • 黑名单：对精度有要求的算子，总是使用float32计算，例如Softmax、BN。 • 其他：按照当前输入类型运算，例如Relu，MaxPool。
O2配置模式	model, optimizer = amp.initialize(model, optimizer, **opt_level="O2"**)	针对全网中float32数据类型的算子，按照内置优化策略，自动将部分float32的算子降低精度到float16，从而在精度损失很小的情况下提升性能并减少内存使用。
O3配置模式	model, optimizer = amp.initialize(model, optimizer, **opt_level="O3"**)	全部算子使用float16计算。
静态Loss Scale功能	model, optimizer = amp.initialize(model, optimizer, opt_level="O2",**loss_scale=128.0**)	静态设置参数确保混合精度训练收敛。
动态Loss Scale功能	model, optimizer = amp.initialize(model, optimizer, opt_level="O2",**loss_scale="dynamic"**)	动态计算Loss Scale值并判断是否溢出。

说明

当前版本的实现方式主要为Python实现，不支持AscendCL或者CUDA优化。

使用混合精度模块

- 使用APEX混合精度模块

  a. 从APEX库中导入AMP。
  ```
  from apex import amp
  ```

  b. 初始化AMP，使其能对模型、优化器以及PyTorch内部函数进行必要的改动。
  ```
  model, optimizer = amp.initialize(model, optimizer, opt_level="O2")
  ```

  更多混合精度模块的使用可参见**官方文档**。

- 使用框架自带AMP功能（ PyTorch 1.8.1版本及以上 ）。
  ```
  model = ...
  optimizer = ...
  #创建缩放器
  for epoch in epochs:
      for input, target in data:
          optimizer.zero_grad()
          with autocast():
              output = model(input)
              loss = loss_fn(output, target)
        ......
  ```

混合精度推理

按混合精度模型初始化后，正常执行模型正向计算即可。

2.5 操作指导

2.5.1 使用步骤

本节使用ResNet50模型为样例，展示在线推理的使用步骤。用户可根据自己的实际情况使用自己所需的模型、数据集、修改代码和参数。

步骤1 生成模型训练权重文件。

打开ModelZoo中**ResNet-PyTorch详情页**，下载模型文件，根据该页面的README完成模型训练，生成权重文件。

步骤2 编辑推理脚本。

创建“resnet50_infer_for_pytorch.py”模型脚本文件，并写入样例代码。单卡场景请写入**单卡场景样例代码**，多卡场景请写入**多卡场景样例代码**。

步骤3 执行推理。

参考**环境变量配置**设置环境变量，请用户自行准备数据集并执行命令进行推理。

- 单卡场景。
  ```
  python3 resnet50_infer_for_pytorch.py \
      --data /data/imagenet \
      --npu 7 \
      --resume ./checkpoint.pth.tar    # ./checkpoint.pth.tar为示例预训练模型文件路径
  ```

- 多卡场景。


```
python3 resnet50_infer_for_pytorch.py --data /data/imagenet/ --resume ./checkpoint.pth.tar --
world_size 1 --rank 0 --amp
# ./checkpoint.pth.tar为示例预训练模型文件路径
```

说明

- 上述为样例输入，用户可根据实际修改传入的参数。
- 在进行推理应用时，应尽量保证应用在生命周期内不频繁初始化。推理模式通过模型**model.eval()**进行设置，并且推理过程要在**with torch.no_grad():**代码分支下运行。

----结束

2.5.2 单卡场景样例

模块和参数设置

引入所需的模块，设置供用户自定义的参数。

```
import argparse
import os
import time
import torch
import torch_npu
import torch.nn.parallel
import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
from apex import amp    # 导入amp模块

model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))


def parse_args():
    """ 用户自定义数据集路径、模型路径 """
    parser = argparse.ArgumentParser(description='PyTorch ImageNet Inferring')
    parser.add_argument('--data', metavar='DIR', default="/data/imagenet",
                        help='path to dataset')
    parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet50',
                        choices=model_names,
                        help='model architecture: ' +
                            ' | '.join(model_names) +
                            ' (default: resnet18)')

    parser.add_argument('-b', '--batch_size', default=512, type=int,
                        metavar='N',
                        help='mini-batch size (default: 256), this is the total '
                            'batch size of all GPUs on the current node when '
                            'using Data Parallel or Distributed Data Parallel')

    parser.add_argument('--resume', default='', type=str, metavar='PATH',
                        help='path to latest checkpoint (default: none)')

    parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                        help='use pre-trained model')

    parser.add_argument('--npu', default=None, type=int,
                        help='NPU id to use.')

    parser.add_argument('-j', '--workers', default=32, type=int, metavar='N',
                        help='number of data loading workers (default: 32)')
```



```
    parser.add_argument('--lr', '--learning_rate', default=0.1, type=float,
                        metavar='LR', help='initial learning rate', dest='lr')
    parser.add_argument('--wd', '--weight_decay', default=1e-4, type=float,
                        metavar='W', help='weight decay (default: 1e-4)',
                        dest='weight_decay')

    args, unknown_args = parser.parse_known_args()
    if len(unknown_args) > 0:
        for bad_arg in unknown_args:
            print("ERROR: Unknown command line arg: %s" % bad_arg)
        raise ValueError("Invalid command line arg(s)")

    return args
......
```

主函数

设置主函数入口。

```
......
def main():
    args = parse_args()
    if args.npu is None:
        args.npu = 0
    global CALCULATE_DEVICE
    CALCULATE_DEVICE = "npu:{}".format(args.npu)
    torch_npu.npu.set_device(CALCULATE_DEVICE)
    print("use ", CALCULATE_DEVICE)
    main_worker(args.npu, args)
......
```

创建模型

在main_worker中创建模型，设置device和优化器。

```
......
def main_worker(npu, args):
    global best_acc1
    args.npu = npu

    print("=> creating model '{}'".format(args.arch))
    model = models.__dict__[args.arch](zero_init_residual=True)

    # 将模型数据复制到昇腾AI处理器中
    model = model.to(CALCULATE_DEVICE)

    optimizer = torch.optim.SGD([
        {'params': [param for name, param in model.named_parameters() if name[-4:] == 'bias'],
'weight_decay': 0.0},
        {'params': [param for name, param in model.named_parameters() if name[-4:] != 'bias'],
         'weight_decay': args.weight_decay}],
        args.lr)
    ......
```

使能混合精度

在main_worker中初始化混合精度模型，使用后可加速运算，但结果的准确率可能会轻微降低。可根据实际场景选择使用。

```
    ......
    model, optimizer = amp.initialize(model, optimizer, opt_level="O2", loss_scale=1024, verbosity=1)
```

加载模型参数

在main_worker中从模型文件中恢复训练好的模型参数并加载。


```
......
if os.path.isfile(args.resume):
    print("=> loading checkpoint '{}'".format(args.resume))
    checkpoint = torch.load(args.resume)

    best_acc1 = checkpoint['best_acc1']
    best_acc1 = best_acc1.to("npu:{}".format(args.npu))

    model.load_state_dict(checkpoint['state_dict'])
    print("=> loaded checkpoint '{}' ".format(args.resume))

else:
    print("=> no checkpoint found at '{}'".format(args.resume))
......
```

初始化数据集

在main_worker中对图像数据进行加载与预处理。

```
......
valdir = os.path.join(args.data, 'val')
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

val_loader = torch.utils.data.DataLoader(
    datasets.ImageFolder(valdir, transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize,
    ])),
    batch_size=args.batch_size, shuffle=True,
    num_workers=args.workers, pin_memory=True)
......
```

运行推理

在main_worker中运行推理。

```
......
validate(val_loader, model, args)
```

在线推理

在线推理的实现代码如下。

```
......
def validate(val_loader, model, args):
    batch_time = AverageMeter('Time', ':6.3f')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, top1, top5],
        prefix='Test: ')

    # ============================================================================
    # 切换到推理模式
    # ============================================================================
    model.eval()

    # ============================================================================
    # 在 torch.no_grad():分支下执行模型正向计算
    # ============================================================================
    with torch.no_grad():
        end = time.time()
```



```
        for i, (images, target) in enumerate(val_loader):

            # 将图像数据置于NPU中
            images = images.to(CALCULATE_DEVICE, non_blocking=True)
            target = target.to(torch.int32).to(CALCULATE_DEVICE, non_blocking=True)

            # 计算输出
            output = model(images)

            # 统计结果精度
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))

            # 测量运行时间
            batch_time.update(time.time() - end)
            end = time.time()

            # 打印推理运算过程日志
            progress.display(i)

        print(' * Acc@1 {top1.avg:.3f} Acc@5 {top5.avg:.3f}'.format(top1=top1, top5=top5))

    return top1.avg


class AverageMeter(object):
    """计算并存储平均值和当前值"""
    def __init__(self, name, fmt=':f'):
        self.name = name
        self.fmt = fmt
        self.reset()
        self.start_count_index = 10

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        if self.count == 0:
            self.batchsize = n

        self.val = val
        self.count += n
        if self.count > (self.start_count_index * self.batchsize):
            self.sum += val * n
            self.avg = self.sum / (self.count - self.start_count_index * self.batchsize)

    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)


class ProgressMeter(object):
    """记录模型运算过程信息"""
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix

    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))

    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
```



```
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'


def accuracy(output, target, topk=(1,)):
    """根据指定值k，计算k个顶部预测的精度"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)

        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))

        res = []
        for k in topk:
            correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res


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

推理完成

当出现推理结果精度的回显时，说明推理完成。样例回显截图如下。

```
.=> loading checkpoint './checkpoint.pth.tar'
....=> loaded checkpoint './checkpoint.pth.tar'
........Test: [ 0/20]   Time 82.844 ( 0.000)    Acc@1   1.37 (  0.00)   Acc@5   4.10 (  0.00)
Test: [ 1/20]   Time  0.038 ( 0.000)    Acc@1   0.20 (  0.00)   Acc@5   1.95 (  0.00)
Test: [ 2/20]   Time  0.131 ( 0.000)    Acc@1   0.78 (  0.00)   Acc@5   4.10 (  0.00)
Test: [ 3/20]   Time  0.144 ( 0.000)    Acc@1   0.78 (  0.00)   Acc@5   2.93 (  0.00)
Test: [ 4/20]   Time  0.164 ( 0.000)    Acc@1   0.98 (  0.00)   Acc@5   3.71 (  0.00)
Test: [ 5/20]   Time  0.109 ( 0.000)    Acc@1   0.20 (  0.00)   Acc@5   3.52 (  0.00)
Test: [ 6/20]   Time  0.110 ( 0.000)    Acc@1   0.00 (  0.00)   Acc@5   3.52 (  0.00)
Test: [ 7/20]   Time  0.127 ( 0.000)    Acc@1   0.78 (  0.00)   Acc@5   3.32 (  0.00)
Test: [ 8/20]   Time  0.149 ( 0.000)    Acc@1   0.39 (  0.00)   Acc@5   2.34 (  0.00)
Test: [ 9/20]   Time  0.128 ( 0.000)    Acc@1   0.39 (  0.00)   Acc@5   2.34 (  0.00)
.Test: [10/20]  Time  1.883 ( 1.883)    Acc@1   0.39 (  0.39)   Acc@5   3.32 (  3.32)
Test: [11/20]   Time  0.616 ( 1.250)    Acc@1   0.20 (  0.29)   Acc@5   2.54 (  2.93)
Test: [12/20]   Time  0.037 ( 0.846)    Acc@1   0.78 (  0.46)   Acc@5   3.12 (  2.99)
Test: [13/20]   Time  0.128 ( 0.666)    Acc@1   0.78 (  0.54)   Acc@5   2.34 (  2.83)
Test: [14/20]   Time  0.129 ( 0.559)    Acc@1   1.17 (  0.66)   Acc@5   2.73 (  2.81)
Test: [15/20]   Time  0.129 ( 0.487)    Acc@1   0.98 (  0.72)   Acc@5   2.54 (  2.77)
Test: [16/20]   Time  0.130 ( 0.436)    Acc@1   0.59 (  0.70)   Acc@5   2.54 (  2.73)
Test: [17/20]   Time  0.146 ( 0.400)    Acc@1   0.39 (  0.66)   Acc@5   2.34 (  2.69)
Test: [18/20]   Time  0.159 ( 0.373)    Acc@1   0.78 (  0.67)   Acc@5   5.08 (  2.95)
........Test: [19/20]  Time 93.696 ( 9.705)    Acc@1   0.74 (  0.68)   Acc@5   3.31 (  2.97)
 * Acc@1 0.676 Acc@5 2.971
```

2.5.3 多卡场景样例

模块和参数设置

引入所需的模块，设置供用户自定义的参数。

```
import argparse
import os
import time
import torch
import torch_npu
import torch.nn.parallel
import torch.multiprocessing as mp
import torch.distributed as dist
import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
```



```
import torchvision.datasets as datasets
import torchvision.models as models
from apex import amp    # 导入amp模块
model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))
def parse_args():
    """ 用户自定义数据集路径、模型路径 """
    parser = argparse.ArgumentParser(description='PyTorch ImageNet Inferring')
    parser.add_argument('--data', metavar='DIR', default="/data/imagenet",
                        help='path to dataset')
    parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet50',
                        choices=model_names,
                        help='model architecture: ' +
                            ' | '.join(model_names) +
                            ' (default: resnet18)')
    parser.add_argument('--epochs', default=100, type=int, metavar='N',
                        help='number of total epochs to run')
    parser.add_argument('-b', '--batch_size', default=512, type=int,
                        metavar='N',
                        help='mini-batch size (default: 256), this is the total '
                            'batch size of all GPUs on the current node when '
                            'using Data Parallel or Distributed Data Parallel')
    parser.add_argument('--resume', default='', type=str, metavar='PATH',
                        help='path to latest checkpoint (default: none)')
    parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                        help='use pre-trained model')
    parser.add_argument('-j', '--workers', default=32, type=int, metavar='N',
                        help='number of data loading workers (default: 32)')
    parser.add_argument('--lr', '--learning_rate', default=0.1, type=float,
                        metavar='LR', help='initial learning rate', dest='lr')
    parser.add_argument('--wd', '--weight_decay', default=1e-4, type=float,
                        metavar='W', help='weight decay (default: 1e-4)',
                        dest='weight_decay')
    parser.add_argument('--addr', default='127.0.0.1', type=str, help='master addr')
    parser.add_argument('--device_list', default='0,1,2,3,4,5,6,7', type=str, help='device id list')
    parser.add_argument('--dist_backend', default='hccl', type=str, help='distributed backend')
    parser.add_argument('--world_size', default=1, type=int,
                        help='number of nodes for distributed training')
    parser.add_argument('--rank', default=0, type=int,
                        help='node rank for distributed training')
    parser.add_argument('--amp', default=False, action='store_true', help='use amp to train the model')
    args, unknown_args = parser.parse_known_args()
    if len(unknown_args) > 0:
        for bad_arg in unknown_args:
            print("ERROR: Unknown command line arg: %s" % bad_arg)
        raise ValueError("Invalid command line arg(s)")
    return args
......
```

主函数

设置主函数入口。

```
......
def main():
    args = parse_args()
    os.environ['MASTER_ADDR'] = args.addr
    os.environ['MASTER_PORT'] = '**'    # **为端口号，请根据实际选择一个闲置端口填写
    args.process_device_map = device_id_to_process_device_map(args.device_list)
    ngpus_per_node = len(args.process_device_map)
    args.world_size = ngpus_per_node * args.world_size
    mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, args))
......
```

创建模型

在main_worker中创建模型，设置device和优化器。


```
......
def main_worker(npu, ngpus_per_node, args):
    global best_acc1
    args.npu = args.process_device_map[npu]
    args.rank = args.rank * ngpus_per_node + npu
    dist.init_process_group(backend=args.dist_backend,
                            world_size=args.world_size, rank=args.rank)
    # ==========================================================================
    # 创建模型
    # ==========================================================================
    print("=> creating model '{}'".format(args.arch))
    model = models.__dict__[args.arch](zero_init_residual=True)
    # 指定推理设备为昇腾AI处理器
    loc = 'npu:{}'.format(args.npu)
    torch_npu.npu.set_device(loc)
    # 计算用于推理的batch_size和workers
    args.batch_size = int(args.batch_size / ngpus_per_node)
    args.workers = int((args.workers + ngpus_per_node - 1) / ngpus_per_node)
    # 将模型数据复制到昇腾AI处理器中
    model = model.to(loc)
    optimizer = torch.optim.SGD([
        {'params': [param for name, param in model.named_parameters() if name[-4:] == 'bias'],
'weight_decay': 0.0},
        {'params': [param for name, param in model.named_parameters() if name[-4:] != 'bias'],
         'weight_decay': args.weight_decay}],
        args.lr)
    ......
```

使能混合精度

在main_worker中初始化混合精度模型，使用后可加速运算，但结果的准确率可能会轻微降低。可根据实际场景选择使用。

```
    ......
    if args.amp:
        model, optimizer = amp.initialize(model, optimizer, opt_level="O2", loss_scale=1024, verbosity=1)
    model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.npu])
    ......
```

加载模型参数

在main_worker中从模型文件中恢复训练好的模型参数并加载。

```
    ......
    if os.path.isfile(args.resume):
        print("=> loading checkpoint '{}'".format(args.resume))
        checkpoint = torch.load(args.resume)
        best_acc1 = checkpoint['best_acc1']
        best_acc1 = best_acc1.to("npu:{}".format(args.npu))
        model.load_state_dict(checkpoint['state_dict'])
        print("=> loaded checkpoint '{}' (epoch {})".format(args.resume, checkpoint['epoch']))
    else:
        print("=> no checkpoint found at '{}'".format(args.resume))
    ......
```

初始化数据集

在main_worker中对图像数据进行加载与预处理。

```
    ......
    valdir = os.path.join(args.data, 'val')
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    val_dataset = datasets.ImageFolder(valdir, transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
```



```
        transforms.ToTensor(),
        normalize,
    ]))
    val_sampler = torch.utils.data.distributed.DistributedSampler(val_dataset, shuffle=False, drop_last=False)
    val_loader = torch.utils.data.DataLoader(
        val_dataset, batch_size=args.batch_size,
        num_workers=args.workers, pin_memory=False, sampler=val_sampler)
    ......
```

运行推理

在main_worker中运行推理。

```
    ......
    validate(val_loader, model, args)
    ......
```

在线推理

在线推理的实现代码如下。

```
......
def validate(val_loader, model, args):
    batch_time = AverageMeter('Time', ':6.3f')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, top1, top5],
        prefix='Test: ')
    # ============================================================================
    # 切换到推理模式
    # ============================================================================
    model.eval()
    # ============================================================================
    # 在 torch.no_grad():分支下执行模型正向计算
    # ============================================================================
    with torch.no_grad():
        end = time.time()
        for i, (images, target) in enumerate(val_loader):
            # 将图像数据置于NPU中
            loc = 'npu:{}'.format(args.npu)
            target = target.to(torch.int32)
            images, target = images.to(loc, non_blocking=False), target.to(loc, non_blocking=False)
            # 计算输出
            output = model(images)
            # 统计结果精度
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))
            # 测量运行时间
            batch_time.update(time.time() - end)
            end = time.time()
            # 打印推理运算过程日志
            progress.display(i)
        print(' * Acc@1 {top1.avg:.3f} Acc@5 {top5.avg:.3f}'.format(top1=top1, top5=top5))
    return top1.avg
class AverageMeter(object):
    """计算并存储平均值和当前值"""
    def __init__(self, name, fmt=':f'):
        self.name = name
        self.fmt = fmt
        self.reset()
        self.start_count_index = 10
    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
```



```
        self.count = 0
    def update(self, val, n=1):
        if self.count == 0:
            self.batchsize = n
        self.val = val
        self.count += n
        if self.count > (self.start_count_index * self.batchsize):
            self.sum += val * n
            self.avg = self.sum / (self.count - self.start_count_index * self.batchsize)
    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)
class ProgressMeter(object):
    """记录模型运算过程信息"""
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix
    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))
    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'
def accuracy(output, target, topk=(1,)):
    """根据指定值k，计算k个顶部预测的精度"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)
        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))
        res = []
        for k in topk:
            correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res
def device_id_to_process_device_map(device_list):
    devices = device_list.split(",")
    devices = [int(x) for x in devices]
    devices.sort()
    process_device_map = dict()
    for process_id, device_id in enumerate(devices):
        process_device_map[process_id] = device_id
    return process_device_map
if __name__ == '__main__':
    main()
```

推理完成

当出现推理结果精度的回显时，说明推理完成。样例回显截图如下。


```
Test: [92/98]   Time  0.665 ( 0.204)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.43)
Test: [93/98]   Time  0.044 ( 0.202)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.43)
Test: [95/98]   Time  0.382 ( 0.201)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.36)
Test: [94/98]   Time  0.045 ( 0.200)   Acc@1   0.00 (  0.00)   Acc@5   3.12 (  0.46)
Test: [96/98]   Time  0.039 ( 0.199)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.36)
Test: [95/98]   Time  0.043 ( 0.198)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.45)
Test: [92/98]   Time  0.587 ( 0.211)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.32)
Test: [93/98]   Time  0.700 ( 0.209)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.48)
Test: [94/98]   Time  0.614 ( 0.204)   Acc@1   0.00 (  0.00)   Acc@5   1.56 (  0.42)
Test: [96/98]   Time  0.391 ( 0.200)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.43)
Test: [93/98]   Time  0.045 ( 0.209)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.32)
Test: [94/98]   Time  0.041 ( 0.207)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.48)
Test: [95/98]   Time  0.041 ( 0.202)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.42)
Test: [94/98]   Time  0.039 ( 0.207)   Acc@1   0.00 (  0.00)   Acc@5   1.56 (  0.33)
Test: [95/98]   Time  0.040 ( 0.205)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.47)
Test: [96/98]   Time  0.044 ( 0.200)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.41)
Test: [95/98]   Time  0.039 ( 0.205)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.33)
Test: [96/98]   Time  0.042 ( 0.203)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.47)
Test: [95/98]   Time  0.616 ( 0.206)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.47)
Test: [96/98]   Time  0.613 ( 0.201)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.41)
Test: [96/98]   Time  0.042 ( 0.204)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.47)
Test: [96/98]   Time  0.499 ( 0.202)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.45)
Test: [96/98]   Time  0.484 ( 0.208)   Acc@1   0.00 (  0.00)   Acc@5   0.00 (  0.32)
Test: [97/98]   Time  0.040 ( 0.200)   Acc@1  14.29 (  0.11)   Acc@5  14.29 (  0.55)
Test: [97/98]   Time  0.032 ( 0.199)   Acc@1  14.29 (  0.11)   Acc@5  14.29 (  0.53)
Test: [97/98]   Time  0.033 ( 0.199)   Acc@1  14.29 (  0.11)   Acc@5  14.29 (  0.52)
Test: [97/98]   Time  0.114 ( 0.198)   Acc@1  16.67 (  0.12)   Acc@5  16.67 (  0.48)
Test: [97/98]   Time  0.036 ( 0.198)   Acc@1  16.67 (  0.12)   Acc@5  16.67 (  0.53)
Test: [97/98]   Time  0.033 ( 0.202)   Acc@1  14.29 (  0.11)   Acc@5  14.29 (  0.57)
Test: [97/98]   Time  0.039 ( 0.206)   Acc@1  14.29 (  0.11)   Acc@5  14.29 (  0.43)
Test: [97/98]   Time  0.320 ( 0.204)   Acc@1  14.29 (  0.11)   Acc@5  14.29 (  0.57)
 * Acc@1 0.107 Acc@5 0.553
 * Acc@1 0.107 Acc@5 0.535
 * Acc@1 0.107 Acc@5 0.517
 * Acc@1 0.107 Acc@5 0.570
 * Acc@1 0.125 Acc@5 0.481
 * Acc@1 0.107 Acc@5 0.428
 * Acc@1 0.125 Acc@5 0.535
 * Acc@1 0.107 Acc@5 0.570
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
THPModule_npu_shutdown success.
```

