[HW3_prob1]_CNN_Training_with_resnet20

October 15, 2022

[5]:
```
import argparse
import os
import time
import shutil

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn



import torchvision
import torchvision.transforms as transforms

from models import *   # bring everything in the folder models

global best_prec
use_gpu = torch.cuda.is_available()
print('=> Building model...')
device = torch.device("cuda" if use_gpu else "cpu")

batch_size = 128

model_name = "resnet20_cifar"
model = resnet20_cifar()



normalize = transforms.Normalize(mean=[0.491, 0.482, 0.447], std=[0.247, 0.243,␣
↪0.262])


train_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
```

```
    download=True,
    transform=transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=2)


test_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        normalize,
    ]))

testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=2)


print_freq = 100 # every 100 batches, accuracy printed. Here, each batch␣
 ↪includes "batch_size" data points
# CIFAR10 has 50,000 training data, and 10,000 validation data.

def train(trainloader, model, criterion, optimizer, epoch):
    batch_time = AverageMeter()   ## at the begining of each epoch, this should␣
 ↪be reset
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    model.train()

    end = time.time()  # measure current time

    for i, (input, target) in enumerate(trainloader):
        # measure data loading time
        data_time.update(time.time() - end)  # data loading time

        input, target = input.cuda(), target.cuda()

        # compute output
        output = model(input)
```

```
        loss = criterion(output, target)

        # measure accuracy and record loss
        prec = accuracy(output, target)[0]
        losses.update(loss.item(), input.size(0))
        top1.update(prec.item(), input.size(0))

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # measure elapsed time
        batch_time.update(time.time() - end) # time spent to process one batch
        end = time.time()


        if i % print_freq == 0:
            print('Epoch: [{0}][{1}/{2}]\t'
                  'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
                  'Data {data_time.val:.3f} ({data_time.avg:.3f})\t'
                  'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
                  'Prec {top1.val:.3f}% ({top1.avg:.3f}%)'.format(
                   epoch, i, len(trainloader), batch_time=batch_time,
                   data_time=data_time, loss=losses, top1=top1))



def validate(val_loader, model, criterion ):
    batch_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    # switch to evaluate mode
    model.eval()

    end = time.time()
    with torch.no_grad():
        for i, (input, target) in enumerate(val_loader):

            input, target = input.cuda(), target.cuda()

            # compute output
            output = model(input)
            loss = criterion(output, target)

            # measure accuracy and record loss
```

```
            prec = accuracy(output, target)[0]
            losses.update(loss.item(), input.size(0))
            top1.update(prec.item(), input.size(0))

            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()

            if i % print_freq == 0:  # This line shows how frequently print out the status. e.g., i%5 => every 5 batch, prints out
                print('Test: [{0}/{1}]\t'
                  'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
                  'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
                  'Prec {top1.val:.3f}% ({top1.avg:.3f}%)'.format(
                   i, len(val_loader), batch_time=batch_time, loss=losses,
                   top1=top1))

    print(' * Prec {top1.avg:.3f}% '.format(top1=top1))
    return top1.avg


def accuracy(output, target, topk=(1,)):
    """Computes the precision@k for the specified values of k"""
    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)
        res.append(correct_k.mul_(100.0 / batch_size))
    return res


class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

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

```
    def update(self, val, n=1):
        self.val = val
        self.sum += val * n    ## n is impact factor
        self.count += n
        self.avg = self.sum / self.count


def save_checkpoint(state, is_best, fdir):
    filepath = os.path.join(fdir, 'checkpoint.pth')
    torch.save(state, filepath)
    if is_best:
        shutil.copyfile(filepath, os.path.join(fdir, 'model_best.pth.tar'))


def adjust_learning_rate(optimizer, epoch):
    """For resnet, the lr starts from 0.1, and is divided by 10 at 80 and 120␣
 ↪epochs"""
    adjust_list = [150, 225]
    if epoch in adjust_list:
        for param_group in optimizer.param_groups:
            param_group['lr'] = param_group['lr'] * 0.1

#model = nn.DataParallel(model).cuda()
#all_params = checkpoint['state_dict']
#model.load_state_dict(all_params, strict=False)
#criterion = nn.CrossEntropyLoss().cuda()
#validate(testloader, model, criterion)
```

```
=> Building model…
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image

def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# get some random training images
dataiter = iter(testloader)
```

```
images, labels = dataiter.next() ## If you run this line, the next data batch␣
↪is called subsequently.

# show images
imshow(torchvision.utils.make_grid(images))
```

[ ]:
```
# This cell is from the website

lr = 4.3e-2
weight_decay = 1e-4
epochs = 200
best_prec = 0

model = model.cuda()
criterion = nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9,␣
↪weight_decay=weight_decay)
# weight decay: for regularization to prevent overfitting


if not os.path.exists('result'):
    os.makedirs('result')

fdir = 'result/'+str(model_name)

if not os.path.exists(fdir):
    os.makedirs(fdir)


for epoch in range(0, epochs):
    adjust_learning_rate(optimizer, epoch)

    train(trainloader, model, criterion, optimizer, epoch)

    # evaluate on test set
    print("Validation starts")
    prec = validate(testloader, model, criterion)

    # remember best precision and save checkpoint
    is_best = prec > best_prec
    best_prec = max(prec,best_prec)
    print('best acc: {:1f}'.format(best_prec))
    save_checkpoint({
        'epoch': epoch + 1,
        'state_dict': model.state_dict(),
        'best_prec': best_prec,
        'optimizer': optimizer.state_dict(),
```

```
    }, is_best, fdir)
```

[6]:
```
fdir = 'result/'+str(model_name)+'/model_best.pth.tar'

checkpoint = torch.load(fdir)
model.load_state_dict(checkpoint['state_dict'])


criterion = nn.CrossEntropyLoss().cuda()

model.eval()
model.cuda()


prec = validate(testloader, model, criterion)
```

```
Test: [0/79]    Time 0.192 (0.192)      Loss 0.2343 (0.2343)    Prec 94.531%
(94.531%)
 * Prec 91.150%
```

[7]:
```
class SaveOutput:
    def __init__(self):
        self.outputs = []
    def __call__(self, module, module_in):
        self.outputs.append(module_in)
    def clear(self):
        self.outputs = []

save_output = SaveOutput()

for layer in model.modules():
    if isinstance(layer, torch.nn.Conv2d):
        print("prehooked")
        layer.register_forward_pre_hook(save_output)

dataiter = iter(trainloader)
images, labels = dataiter.next()
images = images.to(device)
out = model(images)
```

```
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
```

```
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
```

[8]:
```
for layer in model.modules():
    print(layer)
```

```
ResNet_Cifar(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): QuantConv2d(
        16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): QuantConv2d(
        16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
```

```
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): QuantConv2d(
        16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): QuantConv2d(
        16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): QuantConv2d(
          16, 32, kernel_size=(1, 1), stride=(2, 2), bias=False
          (weight_quant): weight_quantize_fn()
        )
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (1): BasicBlock(
      (conv1): QuantConv2d(
        32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): QuantConv2d(
        32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): QuantConv2d(
        32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): QuantConv2d(
```

```
          32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False
          (weight_quant): weight_quantize_fn()
        )
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): QuantConv2d(
        64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): QuantConv2d(
        64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AvgPool2d(kernel_size=8, stride=1, padding=0)
  (fc): Linear(in_features=64, out_features=10, bias=True)
)
Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
Sequential(
  (0): BasicBlock(
    (conv1): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
```

```
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (2): BasicBlock(
    (conv1): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
BasicBlock(
  (conv1): QuantConv2d(
    16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
```

```
  (conv2): QuantConv2d(
    16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
BasicBlock(
  (conv1): QuantConv2d(
    16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
```

```
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
BasicBlock(
  (conv1): QuantConv2d(
    16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
Sequential(
  (0): BasicBlock(
    (conv1): QuantConv2d(
      16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): QuantConv2d(
```

```
        16, 32, kernel_size=(1, 1), stride=(2, 2), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): QuantConv2d(
      32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (2): BasicBlock(
    (conv1): QuantConv2d(
      32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
BasicBlock(
  (conv1): QuantConv2d(
    16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (downsample): Sequential(
    (0): QuantConv2d(
      16, 32, kernel_size=(1, 1), stride=(2, 2), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
QuantConv2d(
  16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
Sequential(
  (0): QuantConv2d(
    16, 32, kernel_size=(1, 1), stride=(2, 2), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  16, 32, kernel_size=(1, 1), stride=(2, 2), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
BasicBlock(
  (conv1): QuantConv2d(
    32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
```

```
  )
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
BasicBlock(
  (conv1): QuantConv2d(
    32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
Sequential(
  (0): BasicBlock(
    (conv1): QuantConv2d(
      32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): QuantConv2d(
        32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): QuantConv2d(
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (2): BasicBlock(
    (conv1): QuantConv2d(
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
```

```
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
BasicBlock(
  (conv1): QuantConv2d(
    32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (downsample): Sequential(
    (0): QuantConv2d(
      32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
QuantConv2d(
  32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
Sequential(
  (0): QuantConv2d(
    32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False
    (weight_quant): weight_quantize_fn()
  )
```

```
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
BasicBlock(
  (conv1): QuantConv2d(
    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
BasicBlock(
  (conv1): QuantConv2d(
    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (conv2): QuantConv2d(
    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
    (weight_quant): weight_quantize_fn()
  )
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
  (relu): ReLU(inplace=True)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
QuantConv2d(
  64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
QuantConv2d(
  64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
weight_quantize_fn()
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
AvgPool2d(kernel_size=8, stride=1, padding=0)
Linear(in_features=64, out_features=10, bias=True)
```

[9]:
```
save_output.outputs[1][0]
```

[9]:
```
tensor([[[[ 1.0605,  1.1399,  0.9017,  …, -1.9879, -1.9879, -1.9879],
          [ 1.1558,  0.9494,  0.5048,  …, -1.9879, -1.9879, -1.9879],
          [ 0.8065,  0.3619,  0.3143,  …, -1.9879, -1.9879, -1.9879],
          …,
          [-0.8447, -0.8130, -0.4954,  …, -1.9879, -1.9879, -1.9879],
          [-0.9559, -0.7177,  0.2031,  …, -1.9879, -1.9879, -1.9879],
          [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879]],

         [[ 0.3404,  0.6147,  0.5663,  …, -1.9835, -1.9835, -1.9835],
          [ 0.4533,  0.4695,  0.2435,  …, -1.9835, -1.9835, -1.9835],
          [ 0.2919,  0.0660,  0.0983,  …, -1.9835, -1.9835, -1.9835],
          …,
          [-0.7086, -0.6602, -0.2568,  …, -1.9835, -1.9835, -1.9835],
          [-0.8700, -0.6118,  0.3565,  …, -1.9835, -1.9835, -1.9835],
          [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835]],

         [[ 0.4642,  0.6887,  0.5391,  …, -1.7061, -1.7061, -1.7061],
          [ 0.5690,  0.5540,  0.2996,  …, -1.7061, -1.7061, -1.7061],
          [ 0.3445,  0.1349,  0.2247,  …, -1.7061, -1.7061, -1.7061],
          …,
          [-0.2842, -0.3141, -0.2093,  …, -1.7061, -1.7061, -1.7061],
          [-0.5985, -0.2991,  0.3894,  …, -1.7061, -1.7061, -1.7061],
          [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061]]],
```

```
[[[-1.1623, -1.1781, -1.1464,  …, -1.9879, -1.9879, -1.9879],
  [-0.8924, -0.9717, -0.8924,  …, -1.9879, -1.9879, -1.9879],
  [-0.5431, -0.5748, -0.5589,  …, -1.9879, -1.9879, -1.9879],
  …,
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879]],

 [[-0.8700, -0.8539, -0.8055,  …, -1.9835, -1.9835, -1.9835],
  [-0.6441, -0.6925, -0.5472,  …, -1.9835, -1.9835, -1.9835],
  [-0.3374, -0.3052, -0.2083,  …, -1.9835, -1.9835, -1.9835],
  …,
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835]],

 [[-1.1373, -1.1373, -1.0924,  …, -1.7061, -1.7061, -1.7061],
  [-1.0176, -1.0625, -0.9427,  …, -1.7061, -1.7061, -1.7061],
  [-0.8080, -0.8080, -0.7332,  …, -1.7061, -1.7061, -1.7061],
  …,
  [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
  [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
  [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061]]],


[[[ 0.4889,  0.5366,  0.6477,  …,  0.6794,  0.6477, -1.9879],
  [ 0.5524,  0.6001,  0.6953,  …,  0.7271,  0.6953, -1.9879],
  [ 0.6159,  0.6794,  0.7747,  …,  0.7906,  0.7430, -1.9879],
  …,
  [ 0.4889,  0.3778,  0.4572,  …, -1.9402, -1.8608, -1.9879],
  [ 0.5207,  0.5207,  0.6001,  …, -1.6544, -1.2258, -1.9879],
  [ 0.4413,  0.5366,  0.6318,  …, -0.4161,  0.0761, -1.9879]],

 [[ 1.2602,  1.3086,  1.3571,  …,  1.3893,  1.3571, -1.9835],
  [ 1.2925,  1.3409,  1.3893,  …,  1.4216,  1.3893, -1.9835],
  [ 1.3248,  1.3732,  1.4216,  …,  1.4539,  1.4055, -1.9835],
  …,
  [ 0.5824,  0.4210,  0.4372,  …, -1.7737, -1.7092, -1.9835],
  [ 0.6147,  0.5663,  0.5824,  …, -1.5317, -1.1121, -1.9835],
  [ 0.5340,  0.5824,  0.6470,  …, -0.2890,  0.1467, -1.9835]],

 [[ 1.8712,  1.9610,  1.9460,  …,  1.9610,  1.9161, -1.7061],
  [ 1.8712,  1.9610,  1.9610,  …,  1.9909,  1.9460, -1.7061],
  [ 1.8862,  1.9610,  1.9610,  …,  2.0209,  1.9760, -1.7061],
  …,
  [ 0.3145,  0.1798,  0.1798,  …, -1.5415, -1.6013, -1.7061],
  [ 0.3894,  0.3595,  0.3295,  …, -1.3768, -1.0775, -1.7061],
```

```
  [ 0.3595,  0.4193,  0.4044,  …, -0.3291,  0.0601, -1.7061]]],


…,


[[[-1.9879, -1.9879, -1.9879,  …,  2.0607,  2.0607,  2.0607],
  [-1.9879, -1.9879, -1.9879,  …,  2.0607,  2.0607,  2.0607],
  [-1.9879, -1.9879, -1.9879,  …,  2.0607,  2.0607,  2.0607],
  …,
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879]],

 [[-1.9835, -1.9835, -1.9835,  …,  2.1317,  2.1317,  2.1317],
  [-1.9835, -1.9835, -1.9835,  …,  2.1317,  2.1317,  2.1317],
  [-1.9835, -1.9835, -1.9835,  …,  2.1317,  2.1317,  2.1317],
  …,
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835]],

 [[-1.7061, -1.7061, -1.7061,  …,  2.1107,  2.1107,  2.1107],
  [-1.7061, -1.7061, -1.7061,  …,  2.1107,  2.1107,  2.1107],
  [-1.7061, -1.7061, -1.7061,  …,  2.1107,  2.1107,  2.1107],
  …,
  [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
  [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
  [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061]]],


[[[-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
  …,
  [-0.1938, -0.2573, -0.1938,  …,  1.3939,  1.2828, -1.9879],
  [-0.2255, -0.1938, -0.1938,  …,  1.4892,  0.3778, -1.9879],
  [-0.3367, -0.1779, -0.1938,  …,  0.6001, -0.0826, -1.9879]],

 [[-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
  …,
  [-0.0470, -0.0954, -0.0954,  …,  1.8089,  1.7121, -1.9835],
  [-0.0792, -0.0470, -0.0631,  …,  1.8896,  0.6470, -1.9835],
  [-0.1922, -0.0308, -0.0631,  …,  0.8891, -0.0147, -1.9835]],
```

```
          [[-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
           [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
           [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
           …,
           [-1.4517, -1.5265, -1.3319,  …,  0.7037,  0.5540, -1.7061],
           [-1.4666, -1.4666, -1.3918,  …,  0.8235, -0.4039, -1.7061],
           [-1.5714, -1.4367, -1.3768,  …, -0.2393, -1.1074, -1.7061]]],


         [[[ 0.3302,  0.8858,  1.3780,  …, -1.9879, -1.9879, -1.9879],
           [ 0.4730,  1.2828,  1.3621,  …, -1.9879, -1.9879, -1.9879],
           [ 1.0922,  1.2828,  1.1716,  …, -1.9879, -1.9879, -1.9879],
           …,
           [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
           [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879],
           [-1.9879, -1.9879, -1.9879,  …, -1.9879, -1.9879, -1.9879]],

          [[ 0.1951,  0.7761,  1.2925,  …, -1.9835, -1.9835, -1.9835],
           [ 0.3404,  1.1795,  1.2602,  …, -1.9835, -1.9835, -1.9835],
           [ 0.9697,  1.1311,  1.0182,  …, -1.9835, -1.9835, -1.9835],
           …,
           [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
           [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835],
           [-1.9835, -1.9835, -1.9835,  …, -1.9835, -1.9835, -1.9835]],

          [[ 0.3445,  0.8534,  1.3174,  …, -1.7061, -1.7061, -1.7061],
           [ 0.4493,  1.1977,  1.2725,  …, -1.7061, -1.7061, -1.7061],
           [ 0.9881,  1.1378,  1.0180,  …, -1.7061, -1.7061, -1.7061],
           …,
           [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
           [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061],
           [-1.7061, -1.7061, -1.7061,  …, -1.7061, -1.7061, -1.7061]]]],
       device='cuda:0')
```

[26]:
```
my_input = save_output.outputs[1][0]
images.size()
```

[26]:
```
torch.Size([128, 3, 32, 32])
```

[33]:
```
#Prehooked BasicBlock0 to BasicBlock1
co = model.layer1
conv_1 = model.layer1[0].conv1
conv_2 = model.layer1[0].conv2
bn_1  = model.layer1[0].bn1
Rel   = model.layer1[0].relu
bn_2 = model.layer1[0].bn2
co
```

[33]:
```
Sequential(
  (0): BasicBlock(
    (conv1): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (2): BasicBlock(
    (conv1): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (conv2): QuantConv2d(
      16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
```

[34]:
```
res = my_input
my_output = Rel((bn_2(conv_2(Rel(bn_1(conv_1(my_input))))))+res)
```

[35]:
```
(my_output - save_output.outputs[3][0]).sum()
```

[35]:
```
tensor(0., device='cuda:0', grad_fn=<SumBackward0>)
```

[ ]:
```

```