project_ResNET

November 28, 2022

[12]:
```
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 *


global best_prec
use_gpu = torch.cuda.is_available()
print('=> Building model...')



batch_size = 128
model_name = "resnet20_quant4bit"
model = resnet20_quant()
#model.conv1 = nn.Conv2d(3, 8, kernel_size=3, stride=1, padding=1, bias=False)
#model.bn1 = bn1 = nn.BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,␣
 ↪track_running_stats=True)
model.layer1[0].conv1 = QuantConv2d(8, 8, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
#model.layer1[0].conv2 = QuantConv2d(8, 8, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
#model.layer1[0].bn1 = nn.BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,␣
 ↪track_running_stats=True)
model.layer1[0].bn2 = nn.Sequential()
```

```
#model.layer1[1].conv1 = QuantConv2d(8, 16, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
print(model)

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()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    model.train()
```

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

        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)
        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
        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 = [80, 120]
    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…
ResNet_Cifar(
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
  (bn1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (layer1): Sequential(
    (0): BasicBlock(
```

```
      (conv1): QuantConv2d(
        8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): Sequential()
    )
    (1): BasicBlock(
      (conv1): QuantConv2d(
        8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): QuantConv2d(
        8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (conv2): QuantConv2d(
        8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (weight_quant): weight_quantize_fn()
      )
      (bn1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (bn2): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): QuantConv2d(
        8, 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(
          8, 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)
)
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
# This cell won't be given, but students will complete the training

lr = 4e-2
weight_decay = 1e-4
epochs = 100
best_prec = 0

#model = nn.DataParallel(model).cuda()
model.cuda()
criterion = nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9,␣
 ↪weight_decay=weight_decay)
#cudnn.benchmark = True

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)
```

[25]:
```
class SaveOutput:
    def __init__(self):
        self.outputs = []
    def __call__(self, module, module_in):
        self.outputs.append(module_in)
    def clear(self):
        self.outputs = []

######### Save inputs from selected layer ##########
save_output = SaveOutput()
device = torch.device("cuda" if use_gpu else "cpu")
for layer in model.modules():
    if isinstance(layer, torch.nn.Conv2d):
        print("prehooked")
        layer.register_forward_pre_hook(save_output)       ## Input for the␣
 ↪module will be grapped
####################################################

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
prehooked
```

[ ]:
```
# HW

#  1. Train with 4 bits for both weight and activation to achieve >90% accuracy
#  2. Find x_int and w_int for the 2nd convolution layer
#  3. Check the recovered psum has similar value to the un-quantized original␣
 ↪psum
#     (such as example 1 in W3S2)
```

[14]:
```
PATH = "result/resnet20_quant4bit/model_best.pth.tar"
checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['state_dict'])
device = torch.device("cuda")

model.cuda()
model.eval()

test_loss = 0
correct = 0

with torch.no_grad():
    for data, target in testloader:
        data, target = data.to(device), target.to(device) # loading to GPU
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(testloader.dataset)

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(
        correct, len(testloader.dataset),
        100. * correct / len(testloader.dataset)))
```

```
Test set: Accuracy: 8987/10000 (90%)
```

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

[26]:
```
torch.Size([128, 8, 32, 32])
```

[222]:
```
w_bit = 4
weight_q = model.layer1[0].conv2.weight_q # quantized value is stored during␣
 ↪the training
w_alpha = model.layer1[0].conv2.weight_quant.wgt_alpha
w_delta = w_alpha/(2**(w_bit-1)-1)
weight_int = weight_q/w_delta
print(weight_int) # you should see clean integer numbers
```

```
tensor([[[[ 0., -1., -1.],
          [-0., -2.,  3.],
          [-0., -2., -3.]],

         [[ 2., -2.,  2.],
          [-2., -2., -0.],
          [-0., -2., -3.]],

         [[-5., -1.,  7.],
          [ 1.,  1.,  7.],
          [-4.,  0.,  3.]],

         [[ 0., -2., -3.],
          [ 0., -3.,  0.],
          [ 0., -1., -1.]],

         [[-2., -2., -3.],
          [ 4., -7., -3.],
          [-0., -2., -3.]],

         [[ 1., -1., -2.],
          [-1.,  3., -2.],
          [ 3., -1., -1.]],

         [[ 1.,  1., -3.],
          [ 0., -5.,  6.],
          [ 1., -2., -0.]],

         [[ 2., -2., -1.],
          [ 1.,  4., -0.],
          [ 1., -0.,  1.]]],


        [[[-1.,  1.,  1.],
          [-1., -1., -1.],
          [-0., -3., -3.]],

         [[-1.,  2.,  1.],
          [ 0.,  1.,  5.],
```

```
  [-1.,  0., -0.]],

 [[-3.,  0., -2.],
  [-1.,  6., -0.],
  [-0.,  4.,  2.]],

 [[ 3.,  2.,  0.],
  [-2., -5., -5.],
  [ 0., -2., -0.]],

 [[ 3.,  2., -0.],
  [ 1.,  6.,  0.],
  [-5.,  4., -0.]],

 [[ 2.,  1., -0.],
  [ 1.,  0.,  1.],
  [ 4.,  0., -2.]],

 [[ 0., -1.,  3.],
  [-0., -6., -4.],
  [-1.,  2.,  2.]],

 [[ 0.,  2.,  1.],
  [ 4.,  2.,  0.],
  [ 0.,  0., -3.]]],


[[[-6.,  1., -0.],
  [ 3., -2., -2.],
  [-0., -2., -2.]],

 [[-1.,  0., -0.],
  [ 3.,  1., -4.],
  [ 2.,  0., -3.]],

 [[-3.,  6.,  2.],
  [ 2., -7.,  2.],
  [ 1., -4., -2.]],

 [[-3., -0.,  3.],
  [-3., -0.,  3.],
  [-2., -1.,  1.]],

 [[-7., -3.,  1.],
  [ 2.,  1., -1.],
  [ 4.,  0.,  0.]],

 [[ 1., -3.,  2.],
```

```
  [-1., -1.,  1.],
  [-1.,  2., -1.]],

 [[-7.,  6.,  3.],
  [-7.,  2.,  1.],
  [ 7., -4., -2.]],

 [[ 3.,  4., -3.],
  [ 2., -1.,  2.],
  [-2., -4., -0.]]],


[[[-0.,  1.,  2.],
  [ 0., -3., -1.],
  [-1., -1.,  0.]],

 [[ 2., -0., -1.],
  [ 1., -2., -3.],
  [ 0.,  1.,  2.]],

 [[ 3., -1.,  1.],
  [ 3.,  0.,  3.],
  [ 1.,  0., -0.]],

 [[-2., -1.,  0.],
  [ 1.,  1.,  2.],
  [ 0.,  1.,  2.]],

 [[ 1.,  1.,  2.],
  [-2., -5., -1.],
  [-1., -1.,  2.]],

 [[ 2., -1.,  3.],
  [-2., -3., -2.],
  [-3., -5., -5.]],

 [[-1.,  0., -1.],
  [ 2., -3.,  2.],
  [ 2.,  2.,  4.]],

 [[ 0., -0.,  3.],
  [ 1.,  6.,  4.],
  [-0.,  4.,  2.]]],


[[[-1.,  0.,  2.],
  [ 2.,  3.,  2.],
  [-1.,  5.,  0.]],
```

```
 [[ 1.,  3.,  2.],
  [ 0.,  5.,  1.],
  [-4.,  1.,  3.]],

 [[-0., -1.,  1.],
  [ 3.,  1.,  1.],
  [-0.,  3.,  1.]],

 [[-3., -0.,  2.],
  [-3., -0.,  1.],
  [-3., -2.,  2.]],

 [[ 0., -1., -2.],
  [ 0.,  0.,  1.],
  [ 1., -1.,  1.]],

 [[ 1., -3.,  0.],
  [-2.,  4., -0.],
  [-2.,  4.,  1.]],

 [[-1.,  2., -1.],
  [ 3.,  2.,  1.],
  [-1.,  1.,  1.]],

 [[ 1., -1., -0.],
  [-1., -3., -1.],
  [-1., -1., -2.]]],


[[[-1., -1.,  1.],
  [-2., -3., -4.],
  [-2., -3.,  1.]],

 [[ 0., -0.,  2.],
  [-0., -2., -3.],
  [ 1., -5., -3.]],

 [[ 0., -1., -4.],
  [ 6.,  3., -7.],
  [ 1.,  1., -1.]],

 [[ 1., -2.,  1.],
  [ 1., -2.,  1.],
  [ 1.,  0., -0.]],

 [[-1.,  2.,  2.],
  [-5.,  6.,  3.],
```

```
  [-3.,  5., -1.]],

 [[-3.,  0., -0.],
  [-1., -1., -0.],
  [-2., -5.,  2.]],

 [[ 1., -5.,  0.],
  [ 3.,  6., -1.],
  [ 3.,  7.,  0.]],

 [[-1.,  2., -2.],
  [-2., -1., -4.],
  [-1., -1.,  0.]]],


[[[ 1.,  4.,  1.],
  [ 6.,  7.,  6.],
  [-4., -2.,  2.]],

 [[ 0.,  1., -0.],
  [ 3.,  4.,  4.],
  [-4., -4., -2.]],

 [[-0.,  2., -1.],
  [ 2.,  0.,  6.],
  [-0., -4., -5.]],

 [[ 1., -1., -2.],
  [ 1.,  3.,  2.],
  [ 0., -1., -4.]],

 [[ 3.,  2., -2.],
  [-7., -1.,  2.],
  [ 2., -1., -4.]],

 [[ 0.,  3., -1.],
  [ 5., -5., -2.],
  [-4.,  4., -2.]],

 [[ 0., -4., -1.],
  [ 1.,  7.,  6.],
  [ 1., -7., -1.]],

 [[-1.,  3.,  2.],
  [-0., -3., -7.],
  [-1.,  3.,  5.]]],
```

```
        [[[ 0.,  0.,  2.],
          [ 2.,  0., -1.],
          [ 1.,  1.,  1.]],

         [[ 1.,  1., -2.],
          [ 1., -0., -2.],
          [-1.,  2.,  0.]],

         [[-1., -0., -3.],
          [-2.,  3., -2.],
          [-0., -1., -1.]],

         [[-1., -4., -3.],
          [-2., -4., -3.],
          [-0., -2., -1.]],

         [[ 1.,  2., -1.],
          [-2.,  7.,  1.],
          [ 1.,  1., -1.]],

         [[ 1.,  3.,  1.],
          [ 0., -6., -2.],
          [-1., -4., -4.]],

         [[ 4., -0.,  1.],
          [ 1.,  5., -2.],
          [ 0., -1., -1.]],

         [[-1.,  3.,  2.],
          [-1.,  3.,  0.],
          [-0.,  3.,  3.]]]], device='cuda:0', grad_fn=<DivBackward0>)
```

[223]:
```
x_bit = 4
x = save_output.outputs[2][0]  # input of the 2nd conv layer
x_alpha  = model.layer1[0].conv2.act_alpha
x_delta = x_alpha/(2**x_bit-1)

act_quant_fn = act_quantization(x_bit) # define the quantization function
x_q = act_quant_fn(x, x_alpha)         # create the quantized value for x

x_int = x_q/x_delta
print(x_int) # you should see clean integer numbers
```

```
tensor([[[[ 0.,  0.,  0.,  …,  2.,  2.,  1.],
          [ 2.,  1.,  1.,  …,  0.,  0.,  0.],
          [ 1.,  0.,  0.,  …,  0.,  0.,  0.],
          …,
          [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
```

```
  [ 1.,  1.,  1.,  …,  1.,  2.,  1.],
  [ 1.,  2.,  2.,  …,  2.,  2.,  1.]],

 [[ 0.,  0.,  0.,  …,  2.,  2.,  2.],
  [ 0.,  1.,  0.,  …,  0.,  0.,  1.],
  [ 0.,  0.,  0.,  …,  1.,  1.,  2.],
  …,
  [ 0.,  0.,  0.,  …,  1.,  1.,  2.],
  [ 0.,  1.,  1.,  …,  1.,  0.,  2.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.]],

 [[ 0.,  0.,  0.,  …,  3.,  2.,  2.],
  [ 0.,  0.,  1.,  …,  2.,  2.,  2.],
  [ 0.,  0.,  1.,  …,  2.,  2.,  2.],
  …,
  [ 6.,  2.,  2.,  …,  2.,  2.,  2.],
  [ 7.,  2.,  2.,  …,  2.,  2.,  1.],
  [ 4.,  3.,  2.,  …,  2.,  2.,  2.]],

 …,

 [[ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 2.,  3.,  3.,  …,  0.,  0.,  0.],
  [ 1.,  1.,  2.,  …,  0.,  0.,  0.],
  …,
  [ 0.,  0.,  0.,  …,  1.,  0.,  0.],
  [ 1.,  1.,  1.,  …,  1.,  1.,  1.],
  [ 0.,  1.,  0.,  …,  0.,  1.,  0.]],

 [[ 2.,  1.,  1.,  …,  3.,  4.,  4.],
  [ 2.,  1.,  1.,  …,  3.,  4.,  4.],
  [ 2.,  1.,  1.,  …,  3.,  4.,  4.],
  …,
  [ 3.,  3.,  2.,  …,  3.,  4.,  4.],
  [ 3.,  4.,  3.,  …,  3.,  4.,  5.],
  [ 4.,  5.,  5.,  …,  5.,  5.,  5.]],

 [[ 3.,  4.,  4.,  …,  1.,  1.,  1.],
  [ 3.,  1.,  2.,  …,  2.,  0.,  1.],
  [ 2.,  2.,  2.,  …,  1.,  0.,  0.],
  …,
  [ 3.,  1.,  1.,  …,  0.,  0.,  0.],
  [ 3.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 2.,  0.,  1.,  …,  1.,  0.,  1.]]],


[[[ 2.,  2.,  2.,  …,  2.,  2.,  1.],
  [ 1.,  1.,  1.,  …,  1.,  0.,  0.],
```

```
 [ 5.,  5.,  5.,  …,  3.,  0.,  0.],
 …,
 [ 1.,  0.,  0.,  …,  0.,  0.,  0.],
 [ 0.,  0.,  0.,  …,  0.,  1.,  1.],
 [ 1.,  1.,  1.,  …,  0.,  2.,  1.]],

[[ 0.,  3.,  2.,  …,  2.,  2.,  2.],
 [ 0.,  0.,  0.,  …,  0.,  0.,  3.],
 [ 0.,  3.,  4.,  …,  3.,  3.,  4.],
 …,
 [ 0.,  1.,  0.,  …,  0.,  2.,  2.],
 [ 0.,  0.,  0.,  …,  0.,  0.,  2.],
 [ 0.,  0.,  0.,  …,  0.,  0.,  0.]],

[[ 6.,  3.,  3.,  …,  3.,  2.,  2.],
 [ 7.,  2.,  2.,  …,  2.,  1.,  2.],
 [ 1.,  4.,  2.,  …,  1.,  8.,  1.],
 …,
 [ 1.,  1.,  0.,  …,  0.,  9.,  2.],
 [ 0.,  2.,  0.,  …,  0., 10.,  1.],
 [ 2.,  2.,  2.,  …,  0.,  6.,  3.]],

…,

[[ 0.,  0.,  0.,  …,  0.,  0.,  0.],
 [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
 [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
 …,
 [ 1.,  1.,  0.,  …,  1.,  0.,  0.],
 [ 2.,  3.,  3.,  …,  2.,  1.,  1.],
 [ 3.,  4.,  3.,  …,  4.,  0.,  1.]],

[[ 4.,  3.,  3.,  …,  3.,  4.,  4.],
 [ 3.,  4.,  3.,  …,  3.,  4.,  4.],
 [ 9., 10., 10.,  …, 12.,  5.,  4.],
 …,
 [ 2.,  1.,  2.,  …,  0.,  6.,  5.],
 [ 1.,  1.,  0.,  …,  0.,  5.,  5.],
 [ 0.,  1.,  1.,  …,  0.,  7.,  6.]],

[[ 3.,  1.,  2.,  …,  2.,  1.,  1.],
 [ 3.,  0.,  0.,  …,  0.,  0.,  1.],
 [ 2.,  0.,  0.,  …,  0.,  0.,  1.],
 …,
 [ 2.,  0.,  2.,  …,  0.,  2.,  0.],
 [ 3.,  0.,  1.,  …,  0.,  2.,  0.],
 [ 2.,  1.,  1.,  …,  2.,  1.,  0.]]],
```

```
[[[ 2.,  2.,  2.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  …,
  [ 0.,  0.,  0.,  …,  6.,  4.,  1.],
  [ 1.,  1.,  2.,  …,  0.,  0.,  0.],
  [ 1.,  2.,  2.,  …,  0.,  0.,  0.]],

 [[ 0.,  3.,  3.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  1.,  2.,  …,  0.,  0.,  0.],
  …,
  [ 0.,  1.,  1.,  …,  0.,  0.,  0.],
  [ 0.,  1.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.]],

 [[ 6.,  3.,  2.,  …,  2.,  1.,  1.],
  [ 5.,  2.,  2.,  …,  1.,  1.,  1.],
  [ 5.,  2.,  2.,  …,  1.,  1.,  2.],
  …,
  [ 5.,  2.,  2.,  …,  3.,  2.,  0.],
  [ 7.,  2.,  2.,  …,  0.,  0.,  0.],
  [ 4.,  3.,  2.,  …,  0.,  0.,  1.]],

 …,

 [[ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 1.,  0.,  0.,  …,  1.,  0.,  1.],
  …,
  [ 1.,  0.,  0.,  …,  1.,  0.,  0.],
  [ 1.,  1.,  1.,  …,  0.,  0.,  2.],
  [ 0.,  1.,  0.,  …,  2.,  1.,  3.]],

 [[ 4.,  3.,  4.,  …,  3.,  3.,  2.],
  [ 3.,  3.,  3.,  …,  3.,  3.,  3.],
  [ 3.,  3.,  4.,  …,  2.,  3.,  2.],
  …,
  [ 3.,  3.,  4.,  …,  5.,  6.,  6.],
  [ 3.,  4.,  4.,  …,  4.,  3.,  1.],
  [ 4.,  5.,  5.,  …,  0.,  0.,  0.]],

 [[ 3.,  1.,  1.,  …,  4.,  4.,  3.],
  [ 3.,  1.,  0.,  …,  3.,  4.,  2.],
  [ 3.,  0.,  0.,  …,  3.,  3.,  2.],
  …,
  [ 3.,  0.,  0.,  …,  0.,  0.,  0.],
```

```
  [ 3.,  0.,  0.,  …,  1.,  2.,  2.],
  [ 2.,  0.,  0.,  …,  4.,  4.,  4.]]],


…,


[[[ 2.,  2.,  2.,  …,  2.,  2.,  1.],
  [ 2.,  2.,  2.,  …,  0.,  0.,  0.],
  [ 6.,  6.,  6.,  …,  0.,  0.,  0.],
  …,
  [ 1.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  2.,  1.],
  [ 0.,  0.,  0.,  …,  0.,  2.,  1.]],

 [[ 0.,  3.,  2.,  …,  2.,  2.,  2.],
  [ 0.,  0.,  0.,  …,  0.,  1.,  1.],
  [ 0.,  2.,  4.,  …,  3.,  3.,  2.],
  …,
  [ 0.,  0.,  0.,  …,  4.,  2.,  2.],
  [ 0.,  0.,  0.,  …,  3.,  1.,  2.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.]],

 [[ 6.,  3.,  3.,  …,  3.,  2.,  2.],
  [ 7.,  2.,  3.,  …,  1.,  3.,  2.],
  [ 0.,  4.,  3.,  …,  8.,  1.,  2.],
  …,
  [ 0.,  0.,  1.,  …, 11.,  2.,  2.],
  [ 0.,  0.,  1.,  …, 12.,  2.,  1.],
  [ 0.,  0.,  1.,  …,  5.,  2.,  2.]],

 …,

 [[ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  …,
  [ 1.,  1.,  2.,  …,  1.,  0.,  0.],
  [ 1.,  2.,  2.,  …,  1.,  0.,  1.],
  [ 3.,  2.,  3.,  …,  1.,  1.,  0.]],

 [[ 4.,  3.,  3.,  …,  3.,  4.,  4.],
  [ 3.,  4.,  4.,  …,  4.,  4.,  4.],
  [10., 11., 12.,  …,  3.,  4.,  4.],
  …,
  [ 2.,  1.,  1.,  …,  5.,  5.,  4.],
  [ 2.,  1.,  1.,  …,  5.,  5.,  5.],
  [ 0.,  0.,  0.,  …,  4.,  5.,  5.]],
```

```
 [[ 3.,  1.,  2.,  …,  2.,  1.,  1.],
  [ 3.,  0.,  0.,  …,  0.,  0.,  1.],
  [ 2.,  0.,  0.,  …,  1.,  1.,  0.],
  …,
  [ 2.,  2.,  2.,  …,  2.,  0.,  0.],
  [ 2.,  2.,  2.,  …,  2.,  0.,  0.],
  [ 4.,  4.,  4.,  …,  2.,  0.,  1.]]],


[[[ 2.,  2.,  2.,  …,  2.,  2.,  1.],
  [ 2.,  2.,  2.,  …,  0.,  0.,  0.],
  [ 6.,  7.,  7.,  …,  0.,  0.,  0.],
  …,
  [ 1.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 1.,  0.,  0.,  …,  0.,  2.,  1.],
  [ 0.,  0.,  0.,  …,  0.,  2.,  1.]],

 [[ 0.,  3.,  2.,  …,  2.,  2.,  2.],
  [ 0.,  0.,  0.,  …,  0.,  1.,  1.],
  [ 0.,  0.,  2.,  …,  2.,  3.,  2.],
  …,
  [ 0.,  0.,  0.,  …,  4.,  2.,  2.],
  [ 0.,  0.,  0.,  …,  3.,  1.,  2.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.]],

 [[ 6.,  3.,  3.,  …,  3.,  2.,  2.],
  [ 7.,  2.,  3.,  …,  1.,  3.,  2.],
  [ 0.,  4.,  3.,  …,  8.,  1.,  2.],
  …,
  [ 0.,  0.,  0.,  …, 11.,  2.,  2.],
  [ 0.,  0.,  1.,  …, 12.,  2.,  1.],
  [ 0.,  0.,  0.,  …,  5.,  2.,  2.]],

 …,

 [[ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  1.,  0.,  …,  1.,  0.,  0.],
  …,
  [ 2.,  2.,  4.,  …,  2.,  0.,  0.],
  [ 2.,  2.,  2.,  …,  1.,  0.,  1.],
  [ 3.,  2.,  4.,  …,  1.,  1.,  0.]],

 [[ 4.,  3.,  3.,  …,  3.,  4.,  4.],
  [ 3.,  4.,  4.,  …,  4.,  4.,  4.],
  [10., 12., 12.,  …,  3.,  4.,  4.],
  …,
```

```
  [ 1.,  0.,  0.,  …,  6.,  5.,  4.],
  [ 2.,  1.,  1.,  …,  5.,  5.,  5.],
  [ 0.,  0.,  0.,  …,  5.,  5.,  5.]],

 [[ 3.,  1.,  2.,  …,  2.,  1.,  1.],
  [ 3.,  0.,  0.,  …,  0.,  0.,  1.],
  [ 2.,  0.,  0.,  …,  1.,  1.,  0.],
  …,
  [ 3.,  1.,  2.,  …,  2.,  0.,  0.],
  [ 2.,  3.,  1.,  …,  3.,  0.,  0.],
  [ 3.,  5.,  5.,  …,  2.,  0.,  1.]]],


[[[ 2.,  2.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  1.,  0.,  0.],
  [ 0.,  1.,  2.,  …,  1.,  1.,  0.],
  …,
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 1.,  1.,  1.,  …,  1.,  2.,  1.],
  [ 1.,  2.,  2.,  …,  2.,  2.,  1.]],

 [[ 0.,  6.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  2.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  3.,  0.,  …,  0.,  0.,  0.],
  …,
  [ 0.,  0.,  0.,  …,  0.,  0.,  2.],
  [ 0.,  1.,  1.,  …,  1.,  0.,  2.],
  [ 0.,  0.,  0.,  …,  0.,  0.,  0.]],

 [[ 6.,  6.,  0.,  …,  2.,  0.,  0.],
  [ 6.,  5.,  0.,  …,  3.,  0.,  0.],
  [ 6.,  6.,  0.,  …,  2.,  2.,  0.],
  …,
  [ 5.,  2.,  2.,  …,  2.,  2.,  2.],
  [ 7.,  2.,  2.,  …,  2.,  2.,  1.],
  [ 4.,  3.,  2.,  …,  2.,  2.,  2.]],

 …,

 [[ 0.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 0.,  0.,  0.,  …,  2.,  3.,  2.],
  [ 1.,  2.,  1.,  …,  1.,  1.,  3.],
  …,
  [ 1.,  0.,  0.,  …,  0.,  0.,  0.],
  [ 1.,  1.,  1.,  …,  1.,  1.,  1.],
  [ 0.,  1.,  0.,  …,  0.,  1.,  0.]],

 [[ 5.,  6.,  1.,  …,  2.,  3.,  2.],
```

```
         [ 4.,  7.,  0.,  …,  4.,  1.,  2.],
         [ 4.,  7.,  2.,  …,  5.,  4.,  1.],
         …,
         [ 3.,  3.,  2.,  …,  3.,  4.,  4.],
         [ 3.,  4.,  3.,  …,  3.,  4.,  5.],
         [ 4.,  5.,  5.,  …,  5.,  5.,  5.]],

        [[ 2.,  0.,  1.,  …,  4.,  4.,  4.],
         [ 2.,  1.,  0.,  …,  2.,  3.,  2.],
         [ 2.,  0.,  0.,  …,  2.,  2.,  1.],
         …,
         [ 3.,  0.,  1.,  …,  1.,  0.,  0.],
         [ 3.,  0.,  0.,  …,  0.,  0.,  0.],
         [ 2.,  0.,  1.,  …,  1.,  0.,  1.]]]], device='cuda:0',
       grad_fn=<DivBackward0>)
```

[224]:
```
conv_int = torch.nn.Conv2d(in_channels = 8, out_channels=8, kernel_size = 3,␣
 ↪padding=1, bias = False)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
relu = nn.ReLU()
bn = nn.BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,␣
 ↪track_running_stats=True).to(device)
output_int = conv_int(x_int)
output_int1 = output_int*w_delta*x_delta+save_output.outputs[1][0]
output_recovered = relu(output_int1)
```

[225]:
```
difference = abs(save_output.outputs[3][0] - output_recovered )
print(difference.mean())  ## It should be small, e.g.,2.3 in my trainned model
```

```
tensor(3.4393e-07, device='cuda:0', grad_fn=<MeanBackward0>)
```

[226]:
```
x_int.size()
```

[226]:
```
torch.Size([128, 8, 32, 32])
```

[227]:
```
X_pad = x_int[0]
print(X_pad.size())
x_test = X_pad[:,0,1:9]
print(x_test)
```

```
torch.Size([8, 32, 32])
tensor([[0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 2.],
        [0., 0., 0., 0., 0., 0., 1., 0.],
        [4., 4., 4., 3., 3., 3., 4., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 1., 1., 1., 2., 1., 2., 0.],
```

```
        [4., 4., 4., 4., 4., 5., 3., 5.]], device='cuda:0',
       grad_fn=<SliceBackward>)
```

[228]:
```
x_pad = torch.zeros(128, 8, 34, 34).cuda()
```

[229]:
```
x_pad[ : ,  :, 1:33, 1:33] = x_int.cuda()
x_new= x_pad[:,:,0:3,0:10]
x_new.size()
```

[229]:
```
torch.Size([128, 8, 3, 10])
```

[230]:
```
X = x_new[0]
X = torch.reshape(X, (X.size(0), -1))
```

[231]:
```
X.size()
```

[231]:
```
torch.Size([8, 30])
```

[234]:
```
bit_precision = 4
file = open('./resnet_out/activation.txt', 'w') #write to file
file.write('#time0row7[msb-lsb],time0row6[msb-lst],....,time0row0[msb-lst]#\n')
file.write('#time1row7[msb-lsb],time1row6[msb-lst],....,time1row0[msb-lst]#\n')
file.write('#................#\n')

for i in range(X.size(1)):  # time step
    for j in range(X.size(0)): # row #
        X_bin = '{0:04b}'.format(int(X[7-j,i].item()+0.001))
        for k in range(bit_precision):
            file.write(X_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close() #close file
```

[235]:
```
X[:,0]
```

[235]:
```
tensor([0., 0., 0., 0., 0., 0., 0., 0.], device='cuda:0',
       grad_fn=<SelectBackward>)
```

[236]:
```
weight_int.size()
```

[236]:
```
torch.Size([8, 8, 3, 3])
```

[237]:
```
weight_int.size() # 8, 8 , 3, 3
W = torch.reshape(weight_int, (weight_int.size(0), weight_int.size(1), -1))
W.size() # 8, 8, 9
```

[237]:
```
torch.Size([8, 8, 9])
```

[238]:
```
bit_precision = 4
#file = open('weight.txt', 'w') #write to file
#file.write('#col0row7[msb-lsb],col0row6[msb-lst],....,col0row0[msb-lst]#\n')
#file.write('#col1row7[msb-lsb],col1row6[msb-lst],....,col1row0[msb-lst]#\n')
#file.write('#................#\n')
for kij in range(9):
    file = open('./resnet_out/w{}.txt'.format(str(kij)), 'w')
    file.write('#col0row7[msb-lsb],col0row6[msb-lst],....,col0row0[msb-lst]#\n')
    file.write('#col1row7[msb-lsb],col1row6[msb-lst],....,col1row0[msb-lst]#\n')
    file.write('#................#\n')
    for i in range(W.size(0)):
        for j in range(W.size(1)):
            if (W[i, 7-j, kij].item()<0):
                W_bin = '{0:04b}'.format(int(W[i,7-j,kij].
 ↪item()+2**bit_precision+0.001))
            else:
                W_bin = '{0:04b}'.format(int(W[i,7-j,kij].item()+0.001))
            for k in range(bit_precision):
                file.write(W_bin[k])
            #file.write(' ')  # for visibility with blank between words, you␣
 ↪can use
        file.write('\n')
    file.close() #close file
```

[239]:
```
W[0,:,0]
```

[239]:
```
tensor([ 0.,  2., -5.,  0., -2.,  1.,  1.,  2.], device='cuda:0',
       grad_fn=<SelectBackward>)
```

[240]:
```
output_int.size()
```

[240]:
```
torch.Size([128, 8, 32, 32])
```

[241]:
```
out = output_int[0]
out.size()
out = torch.reshape(out, (out.size(0), -1))
print(out.size())
out = out [:,0:8]
out.size()
```

```
torch.Size([8, 1024])
```

[241]:
```
torch.Size([8, 8])
```

[242]:
```
bit_precision = 16
file = open('./resnet_out/output.txt', 'w') #write to file
file.write('#time0col7[msb-lsb],time0col6[msb-lst],....,time0col0[msb-lst]#\n')
```

```
file.write('#time1col7[msb-lsb],time1col6[msb-lst],....,time1col0[msb-lst]#\n')
file.write('#................#\n')

for i in range(out.size(1)):
    for j in range(out.size(0)):
        if (out[7-j,i].item()<0):
            O_bin = '{0:016b}'.format(int(out[7-j,i].item()+2**bit_precision+0.
 ↪001))
        else:
            O_bin = '{0:016b}'.format(int(out[7-j,i].item()+0.001))
        for k in range(bit_precision):
            file.write(O_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close()
```

[243]:
```
out
```

[243]:
```
tensor([[-60.,  -4.,  -2.,   0.,  13., -13.,  -7.,   9.],
        [  8.,  37.,  46.,  50.,  38.,  42.,  35.,  21.],
        [-19.,   2.,   5.,   5.,   7.,   7., -22.,  27.],
        [  0.,  -7.,  -8.,  -5.,  -1.,  21.,  -4., -16.],
        [ 19.,   2.,  -2.,  -5.,  -4.,   5.,   1.,  20.],
        [ 32., -16., -15., -16., -21., -26.,  16., -74.],
        [-21., -76., -70., -67., -55., -62., -70., -59.],
        [ 47.,  22.,  19.,  18.,   9.,  15.,   4.,  -3.]], device='cuda:0',
       grad_fn=<SliceBackward>)
```

[244]:
```
x_bit = 4
x0 = save_output.outputs[1][0]  # input of the 2nd conv layer
x0_alpha  = model.layer1[0].conv2.act_alpha
x0_delta = x0_alpha/(2**x_bit-1)

act_quant_fn = act_quantization(x_bit) # define the quantization function
x0_q = act_quant_fn(x0, x0_alpha*w_alpha)         # create the quantized value␣
 ↪for x

x0_int = x0_q/x0_delta/w_delta
print(x0_int)
```

```
tensor([[[[49.0000, 35.0000, 35.0000,  …,  0.0000,  0.0000,  0.0000],
          [14.0000, 14.0000, 14.0000,  …, 21.0000, 21.0000,  0.0000],
          [14.0000, 14.0000, 14.0000,  …, 21.0000, 21.0000,  0.0000],
          …,
          [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
          [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],
```

```
[[ 0.0000,  0.0000,  0.0000,  …, 28.0000, 28.0000, 28.0000],
 [ 0.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
 [ 0.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
 …,
 [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
 [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
 [14.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  7.0000]],

[[49.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
 [49.0000,  7.0000,  7.0000,  …, 14.0000, 14.0000, 49.0000],
 [49.0000,  7.0000,  7.0000,  …, 14.0000, 14.0000, 49.0000],
 …,
 [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
 [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000, 49.0000]],

…,

[[77.0000, 14.0000, 14.0000,  …, 35.0000, 35.0000,  0.0000],
 [70.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
 [70.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
 …,
 [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
 [ 0.0000, 28.0000, 28.0000,  …, 28.0000, 28.0000,  0.0000]],

[[42.0000, 35.0000, 35.0000,  …,  0.0000,  0.0000,  7.0000],
 [14.0000, 14.0000, 14.0000,  …,  0.0000,  0.0000, 14.0000],
 [14.0000, 14.0000, 14.0000,  …,  0.0000,  0.0000, 14.0000],
 …,
 [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
 [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
 [14.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 28.0000]],

[[ 7.0000, 21.0000, 21.0000,  …,  7.0000,  7.0000,  0.0000],
 [ 0.0000,  7.0000,  7.0000,  …, 21.0000, 21.0000,  7.0000],
 [ 0.0000,  7.0000,  7.0000,  …, 21.0000, 21.0000,  7.0000],
 …,
 [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
 [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
 [42.0000, 35.0000, 35.0000,  …, 35.0000, 35.0000, 21.0000]]],


[[[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000, 28.0000,  0.0000],
  …,
```

```
 [14.0000, 21.0000, 21.0000,  …, 28.0000, 14.0000,  0.0000],
 [21.0000, 21.0000, 21.0000,  …, 28.0000, 14.0000,  0.0000],
 [14.0000, 14.0000, 14.0000,  …,  7.0000,  0.0000,  0.0000]],

[[35.0000, 28.0000, 28.0000,  …, 28.0000, 28.0000, 28.0000],
 [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
 [14.0000,  7.0000,  7.0000,  …,  7.0000, 21.0000, 14.0000],
 …,
 [14.0000, 14.0000,  7.0000,  …, 14.0000, 28.0000, 14.0000],
 [14.0000, 14.0000, 14.0000,  …, 14.0000, 28.0000, 14.0000],
 [14.0000, 14.0000, 14.0000,  …,  7.0000, 14.0000,  7.0000]],

[[ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
 [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
 [ 0.0000, 28.0000, 28.0000,  …, 14.0000, 21.0000, 49.0000],
 …,
 [ 7.0000, 14.0000, 21.0000,  …,  0.0000,  0.0000, 49.0000],
 [14.0000, 14.0000, 21.0000,  …,  0.0000,  0.0000, 49.0000],
 [14.0000, 14.0000, 14.0000,  …,  0.0000,  0.0000, 49.0000]],

…,

[[ 0.0000, 35.0000, 35.0000,  …, 35.0000, 35.0000,  0.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
 [ 0.0000, 35.0000, 35.0000,  …, 28.0000, 28.0000,  0.0000],
 …,
 [14.0000, 28.0000, 28.0000,  …, 56.0000,  0.0000,  0.0000],
 [21.0000, 21.0000, 28.0000,  …, 56.0000,  0.0000,  0.0000],
 [21.0000, 14.0000, 14.0000,  …, 63.0000,  0.0000,  0.0000]],

[[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  7.0000],
 [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
 [21.0000, 21.0000, 21.0000,  …, 14.0000,  0.0000, 14.0000],
 …,
 [21.0000, 14.0000, 21.0000,  …,  0.0000,  0.0000, 14.0000],
 [28.0000, 14.0000, 14.0000,  …,  0.0000,  0.0000, 14.0000],
 [21.0000,  7.0000,  7.0000,  …,  0.0000,  7.0000, 28.0000]],

[[21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  0.0000],
 [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
 [42.0000, 42.0000, 42.0000,  …, 42.0000, 35.0000,  7.0000],
 …,
 [14.0000, 14.0000, 21.0000,  …, 28.0000, 35.0000,  7.0000],
 [14.0000, 14.0000, 14.0000,  …, 28.0000, 35.0000,  7.0000],
 [14.0000,  7.0000,  7.0000,  …, 21.0000, 42.0000, 21.0000]]],


[[[ 0.0000,  0.0000,  0.0000,  …, 14.0000, 14.0000, 14.0000],
```

```
 [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000, 14.0000],
 [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000, 14.0000],
 …,
 [21.0000, 21.0000, 21.0000,  …,  0.0000,  0.0000,  0.0000],
 [21.0000, 21.0000, 21.0000,  …, 14.0000, 21.0000, 21.0000],
 [ 0.0000,  0.0000,  0.0000,  …, 28.0000, 28.0000, 21.0000]],

[[35.0000, 28.0000, 28.0000,  …,  7.0000,  7.0000,  7.0000],
 [21.0000,  7.0000,  7.0000,  …,  0.0000,  0.0000,  0.0000],
 [21.0000,  7.0000,  7.0000,  …,  0.0000,  0.0000,  0.0000],
 …,
 [21.0000,  7.0000,  7.0000,  …,  0.0000,  0.0000,  0.0000],
 [21.0000,  7.0000,  7.0000,  …,  0.0000,  0.0000,  0.0000],
 [14.0000,  7.0000,  7.0000,  …,  0.0000,  0.0000,  0.0000]],

[[ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 14.0000],
 [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 21.0000],
 [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 21.0000],
 …,
 [ 0.0000, 14.0000, 14.0000,  …, 21.0000, 21.0000, 35.0000],
 [ 0.0000, 14.0000, 14.0000,  …, 21.0000, 21.0000, 14.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 14.0000, 14.0000,  7.0000]],

…,

[[ 0.0000, 35.0000, 35.0000,  …, 14.0000, 21.0000,  7.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 28.0000, 21.0000, 14.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 28.0000, 21.0000, 14.0000],
 …,
 [ 0.0000, 21.0000, 21.0000,  …, 28.0000, 21.0000, 21.0000],
 [ 0.0000, 21.0000, 21.0000,  …, 14.0000, 14.0000, 28.0000],
 [ 0.0000, 28.0000, 28.0000,  …, 28.0000, 28.0000, 28.0000]],

[[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
 [ 7.0000,  0.0000,  0.0000,  …,  7.0000,  7.0000, 21.0000],
 [ 7.0000,  0.0000,  0.0000,  …,  7.0000,  7.0000, 21.0000],
 …,
 [ 7.0000,  0.0000,  0.0000,  …,  7.0000, 14.0000, 28.0000],
 [ 7.0000,  0.0000,  0.0000,  …, 21.0000, 21.0000, 14.0000],
 [14.0000, 14.0000, 14.0000,  …, 14.0000,  7.0000,  7.0000]],

[[21.0000,  7.0000,  7.0000,  …, 14.0000, 14.0000,  7.0000],
 [35.0000, 21.0000, 21.0000,  …, 14.0000, 14.0000, 14.0000],
 [35.0000, 21.0000, 21.0000,  …, 14.0000, 14.0000, 14.0000],
 …,
 [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000, 21.0000],
 [35.0000, 21.0000, 21.0000,  …, 28.0000, 21.0000, 21.0000],
 [42.0000, 35.0000, 35.0000,  …, 14.0000, 14.0000, 14.0000]]],
```

```
…,


[[[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …, 35.0000, 21.0000,  0.0000],
  …,
  [21.0000, 14.0000, 14.0000,  …, 14.0000, 21.0000,  0.0000],
  [21.0000, 14.0000, 14.0000,  …, 14.0000, 21.0000,  0.0000],
  [84.0000, 77.0000, 77.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[35.0000, 28.0000, 28.0000,  …, 28.0000, 28.0000, 28.0000],
  [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
  [ 7.0000,  7.0000,  7.0000,  …, 21.0000,  7.0000, 14.0000],
  …,
  [ 0.0000,  7.0000,  7.0000,  …, 35.0000,  7.0000, 14.0000],
  [ 0.0000,  7.0000,  7.0000,  …, 35.0000,  7.0000, 14.0000],
  [ 0.0000,  7.0000,  7.0000,  …, 21.0000,  7.0000,  7.0000]],

 [[ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
  [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
  [ 0.0000, 28.0000, 28.0000,  …, 21.0000, 14.0000, 49.0000],
  …,
  [49.0000,  7.0000,  7.0000,  …,  0.0000, 14.0000, 49.0000],
  [49.0000, 14.0000, 14.0000,  …,  0.0000, 14.0000, 49.0000],
  [42.0000,  7.0000,  0.0000,  …,  0.0000, 21.0000, 49.0000]],

 …,

 [[ 0.0000, 35.0000, 35.0000,  …, 35.0000, 35.0000,  0.0000],
  [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000, 35.0000, 42.0000,  …, 35.0000, 21.0000,  0.0000],
  …,
  [70.0000, 21.0000, 21.0000,  …,  0.0000, 21.0000,  0.0000],
  [70.0000, 21.0000, 21.0000,  …,  0.0000, 21.0000,  0.0000],
  [63.0000, 14.0000, 14.0000,  …,  0.0000, 28.0000,  0.0000]],

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  7.0000],
  [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
  [28.0000, 21.0000, 21.0000,  …,  0.0000,  0.0000, 14.0000],
  …,
  [21.0000, 14.0000, 14.0000,  …,  0.0000,  0.0000, 14.0000],
  [21.0000, 14.0000, 14.0000,  …,  0.0000,  0.0000, 14.0000],
  [ 7.0000,  7.0000,  7.0000,  …,  7.0000, 14.0000, 28.0000]],

 [[21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  0.0000],
```

```
  [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
  [49.0000, 49.0000, 49.0000,  …, 42.0000, 21.0000,  7.0000],
  …,
  [ 0.0000,  7.0000,  7.0000,  …, 56.0000, 21.0000,  7.0000],
  [ 0.0000,  7.0000,  7.0000,  …, 56.0000, 21.0000,  7.0000],
  [ 0.0000,  0.0000,  0.0000,  …, 49.0000, 35.0000, 21.0000]]],


[[[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …, 35.0000, 21.0000,  0.0000],
  …,
  [21.0000, 21.0000, 28.0000,  …, 14.0000, 21.0000,  0.0000],
  [21.0000, 14.0000, 21.0000,  …, 14.0000, 21.0000,  0.0000],
  [91.0000, 84.0000, 84.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[35.0000, 28.0000, 28.0000,  …, 28.0000, 28.0000, 28.0000],
  [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
  [ 7.0000,  7.0000,  7.0000,  …, 21.0000,  7.0000, 14.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …, 35.0000,  7.0000, 14.0000],
  [ 0.0000,  7.0000,  0.0000,  …, 35.0000,  7.0000, 14.0000],
  [ 0.0000,  7.0000,  7.0000,  …, 21.0000,  7.0000,  7.0000]],

 [[ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
  [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
  [ 0.0000, 28.0000, 28.0000,  …, 14.0000, 14.0000, 49.0000],
  …,
  [49.0000,  7.0000,  0.0000,  …,  0.0000, 14.0000, 49.0000],
  [49.0000,  7.0000,  7.0000,  …,  0.0000, 14.0000, 49.0000],
  [42.0000,  0.0000,  0.0000,  …,  0.0000, 21.0000, 49.0000]],

 …,

 [[ 0.0000, 35.0000, 35.0000,  …, 35.0000, 35.0000,  0.0000],
  [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000, 35.0000, 35.0000,  …, 28.0000, 21.0000,  0.0000],
  …,
  [70.0000, 21.0000, 42.0000,  …,  0.0000, 21.0000,  0.0000],
  [77.0000, 21.0000, 14.0000,  …,  0.0000, 21.0000,  0.0000],
  [70.0000, 14.0000, 21.0000,  …,  0.0000, 28.0000,  0.0000]],

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  7.0000],
  [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
  [21.0000, 21.0000, 21.0000,  …,  7.0000,  0.0000, 14.0000],
  …,
  [14.0000, 21.0000, 21.0000,  …,  7.0000,  0.0000, 14.0000],
  [14.0000, 14.0000, 21.0000,  …,  7.0000,  0.0000, 14.0000],
```

```
  [ 7.0000,  7.0000,  7.0000,  …, 14.0000, 14.0000, 28.0000]],

 [[21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  0.0000],
  [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
  [49.0000, 49.0000, 49.0000,  …, 42.0000, 21.0000,  7.0000],
  …,
  [ 0.0000,  7.0000, 21.0000,  …, 56.0000, 21.0000,  7.0000],
  [ 0.0000,  7.0000,  7.0000,  …, 56.0000, 21.0000,  7.0000],
  [ 0.0000,  0.0000,  0.0000,  …, 49.0000, 35.0000, 21.0000]]],


[[[ 0.0000,  0.0000, 21.0000,  …, 14.0000, 14.0000, 21.0000],
  [21.0000,  0.0000, 28.0000,  …, 21.0000, 21.0000, 21.0000],
  [21.0000,  0.0000, 14.0000,  …,  7.0000, 21.0000, 28.0000],
  …,
  [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [21.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[35.0000, 28.0000,  0.0000,  …,  7.0000,  0.0000,  0.0000],
  [21.0000, 14.0000,  0.0000,  …,  7.0000,  0.0000,  0.0000],
  [21.0000, 14.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  …,
  [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
  [21.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000, 14.0000],
  [14.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  7.0000]],

 [[ 0.0000, 49.0000, 49.0000,  …, 14.0000, 14.0000, 14.0000],
  [ 0.0000, 49.0000, 49.0000,  …, 14.0000, 21.0000,  7.0000],
  [ 0.0000, 49.0000, 49.0000,  …,  7.0000, 14.0000, 14.0000],
  …,
  [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
  [ 0.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 49.0000],
  [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000, 49.0000]],

 …,

 [[ 0.0000,  0.0000, 91.0000,  …, 21.0000, 28.0000, 28.0000],
  [ 0.0000,  0.0000, 77.0000,  …, 14.0000, 28.0000, 21.0000],
  [ 0.0000,  0.0000, 70.0000,  …,  7.0000, 14.0000, 28.0000],
  …,
  [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  0.0000],
  [ 0.0000, 28.0000, 28.0000,  …, 28.0000, 28.0000,  0.0000]],

 [[ 0.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  7.0000],
  [ 7.0000, 14.0000,  0.0000,  …,  7.0000,  7.0000,  7.0000],
  [ 7.0000,  7.0000,  0.0000,  …,  7.0000,  7.0000, 14.0000],
```

```
          …,
          [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
          [ 7.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 14.0000],
          [14.0000, 14.0000, 14.0000,  …, 14.0000, 14.0000, 28.0000]],

         [[21.0000,  0.0000,  0.0000,  …, 14.0000, 14.0000, 14.0000],
          [35.0000,  7.0000,  0.0000,  …, 14.0000, 14.0000, 14.0000],
          [35.0000,  7.0000,  0.0000,  …, 21.0000, 14.0000, 14.0000],
          …,
          [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
          [35.0000, 21.0000, 21.0000,  …, 21.0000, 21.0000,  7.0000],
          [42.0000, 35.0000, 35.0000,  …, 35.0000, 35.0000, 21.0000]]]],
       device='cuda:0', grad_fn=<DivBackward0>)
```

[245]:
```
conv_int = torch.nn.Conv2d(in_channels = 8, out_channels=8, kernel_size = 3,␣
 ↪padding=1, bias = False)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
relu = nn.ReLU()
bn = nn.BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,␣
 ↪track_running_stats=True).to(device)
output_int = conv_int(x_int)
output_int = (output_int+x0_int)*w_delta*x_delta
output_recovered = relu(output_int)
print(conv_int(x_int))
```

```
tensor([[[[ -60.,   -4.,   -2.,  …,   -9.,  -31.,  -65.],
          [ -45.,  -34.,  -26.,  …,  -24.,  -63.,  -84.],
          [ -48.,   -1.,   -6.,  …,  -34.,  -64.,  -97.],
          …,
          [  -8.,  -66.,  -24.,  …,  -14.,  -66.,  -95.],
          [  10.,  -47.,  -19.,  …,  -15.,  -60.,  -94.],
          [  18.,   -8.,   13.,  …,    1.,  -26.,  -71.]],

         [[   8.,   37.,   46.,  …,    9.,    4.,   35.],
          [  19.,   45.,   47.,  …,   29.,   42.,   55.],
          [  20.,   25.,   48.,  …,   27.,   28.,   49.],
          …,
          [  92.,   47.,   45.,  …,   19.,   17.,   52.],
          [  64.,    6.,    7.,  …,  -16.,    0.,   37.],
          [   3.,  -31.,  -25.,  …,  -20.,  -14.,   27.]],

         [[ -19.,    2.,    5.,  …,  -19.,  -23.,   -5.],
          [ -21.,  -33.,  -28.,  …,  -15.,  -33.,  -43.],
          [  15.,   -5.,  -11.,  …,  -23.,  -21.,  -28.],
          …,
          [  20.,  -19.,  -40.,  …,    9.,  -15.,  -43.],
          [ -38.,  -31.,  -41.,  …,  -13.,  -14.,  -35.],
          [  72.,  -12.,  -26.,  …,   -5.,   -1.,  -56.]],
```

```
 …,

 [[  32.,  -16.,  -15.,  …,   48.,   52.,  113.],
  [  12.,  -10.,  -10.,  …,   40.,   49.,  114.],
  [  30.,  -19.,  -33.,  …,   39.,   56.,  118.],
  …,
  [  19.,   38.,    2.,  …,   29.,   55.,  106.],
  [  24.,   73.,   42.,  …,   45.,   55.,  123.],
  [ -11.,   14.,   10.,  …,    9.,   27.,   60.]],

 [[ -21.,  -76.,  -70.,  …,   77.,   52.,   16.],
  [  15.,   39.,   28.,  …,  -20.,  -19.,  -13.],
  [  11.,    1.,  -25.,  …,   17.,   -9.,   -8.],
  …,
  [   5.,    1.,    4.,  …,   14.,   -8.,  -19.],
  [  -1.,   32.,   19.,  …,   17.,   18.,   16.],
  [ 100.,   95.,  108.,  …,   99.,   89.,   50.]],

 [[  47.,   22.,   19.,  …,    4.,    8.,   60.],
  [  23.,   24.,   21.,  …,   18.,   15.,   92.],
  [  44.,   37.,   22.,  …,   16.,   14.,   90.],
  …,
  [  37.,    9.,   24.,  …,   -4.,   -2.,   85.],
  [  21.,   -2.,   14.,  …,   -1.,    1.,   84.],
  [  37.,   29.,   52.,  …,   47.,   40.,  105.]]],


[[[   3.,  -23.,    3.,  …,   -3.,  -39.,  -65.],
  [ -28.,  -82.,  -82.,  …,  -67.,  -76., -116.],
  [  23.,  -13.,   -4.,  …,  -19.,  -89., -126.],
  …,
  [  -7.,  -17.,  -66.,  …,  156.,  -72., -152.],
  [ -23.,  -61.,  -29.,  …,  147.,  -49., -125.],
  [  -6.,  -11.,  -31.,  …,  145.,  -43.,  -96.]],

 [[  58.,   17.,    3.,  …,    5.,    5.,   35.],
  [  53.,   15.,   10.,  …,   30.,   46.,   44.],
  [ -87., -127., -136.,  …,  -83.,  112.,   57.],
  …,
  [  26.,   42.,   26.,  …,   17.,  118.,   20.],
  [  24.,   73.,   47.,  …,    7.,  130.,   10.],
  [  36.,   52.,   53.,  …,   -8.,   27.,   -1.]],

 [[-109.,  -24.,  -10.,  …,  -13.,  -27.,   -8.],
  [ -57.,  -70.,  -63.,  …,  -43.,  -10.,  -27.],
  [  79.,  -62.,  -52.,  …,  -46., -235.,    2.],
  …,
```

```
  [  -4.,   47.,   74.,  …,   15.,  -28.,  -57.],
  [  10.,  -43.,  -21.,  …,   23.,  -46.,  -35.],
  [   9.,   -4.,  -36.,  …,   13.,   91.,  -89.]],

 …,

 [[  34.,   57.,   25.,  …,   37.,   47.,  103.],
  [  41.,   84.,   66.,  …,   97.,   75.,  102.],
  [ -39.,  -28.,    2.,  …,  -27.,   84.,  164.],
  …,
  [  14.,   -9.,  -13.,  …, -150.,   66.,  158.],
  [  -1.,   -3.,   -7.,  …, -165.,   72.,  168.],
  [  -1.,   -8.,    1.,  …, -123.,   20.,   86.]],

 [[  38.,   56.,   57.,  …,   62.,   48.,    8.],
  [ -45.,  -60.,  -74.,  …, -137.,  -40.,  -12.],
  [ 248.,  293.,  315.,  …,  245.,  -32.,  -14.],
  …,
  [  27.,  -11.,  -21.,  …,   15.,   -6.,    6.],
  [ -12.,  -56.,  -51.,  …,   51.,  -48.,   15.],
  [   2.,   29.,   23.,  …,   54.,  147.,   54.]],

 [[  29.,   -2.,    1.,  …,   -2.,    4.,   64.],
  [  19.,   -4.,    8.,  …,   -6.,    9.,   99.],
  [  12.,   13.,   13.,  …,   31.,   40.,   80.],
  …,
  [   6.,  -18.,   31.,  …,  -88.,   40.,   75.],
  [ -10.,   -4.,  -21.,  …,  -95.,   25.,   56.],
  [  10.,   16.,   22.,  …,  -60.,   67.,   83.]]],


[[[   4.,  -14.,  -17.,  …,    6.,   11.,   -3.],
  [ -10.,  -61.,  -55.,  …,  -21.,    0.,  -16.],
  [  12.,  -42.,  -71.,  …,    0.,   -7.,   -2.],
  …,
  [  13.,  -50.,  -78.,  …,   -1.,  -46.,  -61.],
  [  17.,  -41.,  -73.,  …,  -49.,  -57.,   18.],
  [  18.,  -23.,  -36.,  …,  -12.,    0.,   33.]],

 [[  55.,   21.,    8.,  …,   11.,   11.,   33.],
  [  73.,   54.,   55.,  …,   22.,   29.,   34.],
  [  55.,   19.,   23.,  …,   26.,   28.,   31.],
  …,
  [  54.,    9.,   13.,  …,  -51.,  -50.,  -49.],
  [  61.,   -1.,   22.,  …,  -21.,    4.,    7.],
  [   3.,  -28.,  -29.,  …,   63.,   66.,   50.]],

 [[ -95.,  -25.,  -28.,  …,  -10.,  -20.,  -22.],
```

```
  [  12.,  -34.,  -41.,  …,   10.,  -40.,  -23.],
  [ -16.,  -29.,  -20.,  …,  -26.,  -11.,  -26.],
  …,
  [ -24.,   -4.,  -13.,  …,  -81.,   32.,   27.],
  [ -36.,    3.,  -10.,  …,  -22.,  -41.,  -88.],
  [  72.,   -8.,   15.,  …,   18.,   -4.,  -34.]],

 …,

 [[  38.,   59.,    3.,  …,   47.,   34.,   51.],
  [  21.,   55.,   27.,  …,    2.,   14.,   44.],
  [   9.,   48.,   41.,  …,   -9.,    5.,   30.],
  …,
  [  10.,   55.,   49.,  …,   34.,   99.,   29.],
  [  24.,   72.,   55.,  …,  -18.,  -17.,  -61.],
  [ -11.,   23.,   60.,  …,  -37.,  -45.,  -25.]],

 [[  47.,   78.,   75.,  …,   -7.,  -15.,  -24.],
  [ -10.,  -12.,  -33.,  …,    8.,   -4.,   12.],
  [  -2.,   17.,  -20.,  …,   -2.,   15.,   -9.],
  …,
  [ -12.,    3.,  -19.,  …,  183.,  179.,   92.],
  [  -6.,   18.,  -17.,  …,   73.,   -1.,  -16.],
  [ 100.,  100.,  142.,  …,  -73.,  -45.,  -20.]],

 [[  33.,   -2.,  -19.,  …,   42.,   33.,   31.],
  [  29.,    4.,  -22.,  …,   44.,   42.,   45.],
  [  19.,  -13.,  -21.,  …,   28.,   39.,   33.],
  …,
  [   8.,  -22.,  -37.,  …,   26.,   44.,   60.],
  [  14.,  -22.,  -35.,  …,   59.,   51.,   15.],
  [  37.,   31.,   38.,  …,   22.,   35.,   13.]]],


…,


[[[   1.,  -23.,   -1.,  …,   -4.,  -31.,  -69.],
  [ -30.,  -61.,  -91.,  …,  -46., -100.,  -76.],
  [  21.,    2.,   -1.,  …,  -28., -104., -108.],
  …,
  [ -52.,  -14.,  -12.,  …,  -47., -141.,  -96.],
  [ -53.,  -15.,  -21.,  …,   -8.,  -99.,  -92.],
  [ -34.,  -19.,  -15.,  …,   -4.,  -67.,  -69.]],

 [[  52.,   11.,   12.,  …,   14.,   13.,   29.],
  [  43.,    1.,    4.,  …,   36.,   40.,   53.],
  [-117., -168., -174.,  …,  112.,   35.,   44.],
```

```
 …,
  [  11.,   37.,   48.,  …,  154.,  -18.,   45.],
  [   0.,   26.,   28.,  …,  139.,  -21.,   35.],
  [  40.,   77.,   87.,  …,   58.,  -32.,   24.]],

 [[-113.,  -32.,  -30.,  …,  -19.,  -18.,    2.],
  [ -63.,  -78.,  -88.,  …,   22.,  -39.,  -35.],
  [  96.,  -60.,  -47.,  …, -258.,    5.,  -10.],
  …,
  [  -1.,  -15.,  -10.,  …,  -57.,  -32.,  -45.],
  [   8.,  -35.,  -12.,  …,  -30.,  -48.,  -46.],
  [  19.,   11.,   16.,  …,  102.,  -35.,  -65.]],

 …,

 [[  33.,   47.,   31.,  …,   43.,   56.,  112.],
  [  38.,   71.,   74.,  …,   70.,   51.,  115.],
  [ -38.,  -48.,   -9.,  …,   55.,  112.,  113.],
  …,
  [  26.,   -7.,  -15.,  …,   31.,  123.,  116.],
  [   3.,  -17.,  -32.,  …,   22.,  112.,  128.],
  [   1.,  -22.,  -33.,  …,  -23.,   38.,   64.]],

 [[  42.,   47.,   34.,  …,   44.,   47.,   14.],
  [ -41.,  -50.,  -60.,  …,    1.,  -24.,  -24.],
  [ 283.,  346.,  373.,  …,  -51.,  -24.,   -9.],
  …,
  [   0.,   -8.,  -24.,  …,    7.,   17.,  -14.],
  [  25.,   -1.,    8.,  …,   31.,   45.,   15.],
  [ -55.,  -44.,  -58.,  …,  107.,   85.,   50.]],

 [[  32.,   -2.,   -1.,  …,   -3.,    7.,   60.],
  [  15.,  -21.,    8.,  …,   24.,   35.,   87.],
  [  10.,    2.,   -4.,  …,   38.,   10.,   94.],
  …,
  [  33.,   27.,   17.,  …,   23.,   -8.,   92.],
  [  29.,   27.,   21.,  …,   24.,  -17.,   91.],
  [  27.,   36.,   26.,  …,   37.,   25.,  109.]]],


[[[   1.,  -23.,   -1.,  …,   -3.,  -31.,  -69.],
  [ -28.,  -59.,  -78.,  …,  -40., -101.,  -76.],
  [  32.,    9.,    8.,  …,  -17., -100., -108.],
  …,
  [ -30.,  -15.,   39.,  …,  -58., -138.,  -96.],
  [ -45.,   -3.,  -36.,  …,   -8.,  -97.,  -92.],
  [ -43.,  -22.,    7.,  …,  -10.,  -65.,  -69.]],
```

```
[[  52.,   11.,   12.,  …,    8.,   18.,   29.],
 [  40.,   -3.,    6.,  …,   21.,   42.,   53.],
 [-137., -187., -179.,  …,  114.,   35.,   44.],
 …,
 [  10.,   53.,   30.,  …,  154.,  -13.,   45.],
 [ -16.,    2.,   24.,  …,  138.,  -13.,   35.],
 [  43.,   81.,   78.,  …,   55.,  -32.,   24.]],

[[-113.,  -32.,  -30.,  …,  -20.,  -24.,    2.],
 [ -62.,  -78.,  -83.,  …,   30.,  -47.,  -35.],
 [ 103.,  -59.,  -63.,  …, -272.,   12.,  -10.],
 …,
 [ -27.,   14.,   98.,  …,  -52.,  -54.,  -45.],
 [  13.,  -33.,  -16.,  …,  -33.,  -52.,  -46.],
 [  19.,   12.,   25.,  …,  110.,  -39.,  -65.]],

…,

[[  33.,   47.,   31.,  …,   38.,   60.,  112.],
 [  47.,   87.,   84.,  …,   60.,   54.,  115.],
 [ -39.,  -46.,   -5.,  …,   57.,  116.,  113.],
 …,
 [   3.,  -34.,  -66.,  …,   39.,  119.,  116.],
 [  -5.,  -25.,  -33.,  …,   29.,  110.,  128.],
 [  -8.,  -24.,  -48.,  …,  -13.,   40.,   64.]],

[[  42.,   47.,   34.,  …,   44.,   45.,   14.],
 [ -38.,  -41.,  -45.,  …,   24.,  -16.,  -24.],
 [ 288.,  352.,  361.,  …,  -82.,  -23.,   -9.],
 …,
 [  -3.,  -27.,  -66.,  …,    8.,   18.,  -14.],
 [  35.,   35.,   27.,  …,   19.,   49.,   15.],
 [ -51.,  -64.,  -72.,  …,  112.,   85.,   50.]],

[[  32.,   -2.,   -1.,  …,   -6.,    6.,   60.],
 [  11.,  -28.,    8.,  …,   10.,   35.,   87.],
 [   5.,   -1.,   -3.,  …,   22.,    7.,   94.],
 …,
 [  13.,   -9.,  -46.,  …,   34.,   -4.,   92.],
 [  19.,   20.,   17.,  …,   30.,  -12.,   91.],
 [  23.,   34.,    6.,  …,   43.,   25.,  109.]]],


[[[ -26., -171.,   41.,  …,  -12.,   -4.,   25.],
  [ -45., -255.,  -27.,  …,  -93.,  -17.,   -6.],
  [ -49., -219.,  -13.,  …,  -30.,  -60.,   12.],
  …,
  [  -8.,  -50.,   -8.,  …,  -19.,  -66.,  -94.],
```

```
   [  13.,  -37.,  -21.,  …,  -18.,  -57.,  -96.],
   [  18.,   -8.,   13.,  …,    1.,  -26.,  -71.]],

  [[  75.,  132.,  -98.,  …,   13.,  -21.,   19.],
   [  79.,  168.,  -33.,  …,   55.,   19.,   13.],
   [  69.,  135.,  -77.,  …,   12.,   51.,   21.],
   …,
   [  72.,   42.,   45.,  …,   33.,   34.,   51.],
   [  62.,    6.,    7.,  …,  -13.,   -5.,   35.],
   [   3.,  -31.,  -25.,  …,  -20.,  -14.,   27.]],

  [[-132.,  -38.,  133.,  …,  -18.,   32.,  -32.],
   [  -3.,  -77.,  -70.,  …,  -69.,  -23.,   11.],
   [ -15.,  -65.,  -63.,  …,  -48., -117.,    3.],
   …,
   [  -1.,    6.,   18.,  …,  -24.,  -59.,  -79.],
   [ -41.,  -24.,  -44.,  …,   -2.,  -11.,  -31.],
   [  72.,  -12.,  -26.,  …,   -5.,   -1.,  -56.]],

  …,

  [[  31.,  257.,  -34.,  …,   70.,    1.,   -8.],
   [  38.,  250.,  -45.,  …,   80.,   23.,  -10.],
   [  20.,  236.,  -35.,  …,   15.,   72.,   14.],
   …,
   [  13.,   48.,    3.,  …,   13.,   41.,   88.],
   [  25.,   69.,   43.,  …,   43.,   59.,  122.],
   [ -11.,   14.,   10.,  …,    9.,   27.,   60.]],

  [[  43.,    8.,    7.,  …,  -42.,   -7.,  -26.],
   [ -47.,  -21.,  -49.,  …,  -25.,  -27.,   35.],
   [ -31.,    7.,   41.,  …,   63.,   -1.,  -27.],
   …,
   [ -16.,  -13.,   -6.,  …,   12.,   12.,   -1.],
   [  -4.,   26.,   20.,  …,   20.,   17.,   15.],
   [ 100.,   95.,  108.,  …,   99.,   89.,   50.]],

  [[   5.,  121.,   -2.,  …,    1.,    6.,   -2.],
   [ -31.,  155.,   13.,  …,   23.,  -26.,    3.],
   [ -44.,  124.,    0.,  …,   53.,   48.,   -8.],
   …,
   [  25.,   11.,   25.,  …,   11.,   14.,  100.],
   [  20.,   -4.,   15.,  …,   -1.,    1.,   84.],
   [  37.,   29.,   52.,  …,   47.,   40.,  105.]]]], device='cuda:0',
grad_fn=<CudnnConvolutionBackward>)
```

[246]:
```
difference = abs(save_output.outputs[3][0] - output_recovered )
print(difference.mean())  ## It should be small, e.g.,2.3 in my trainned model
```

```
tensor(0.1251, device='cuda:0', grad_fn=<MeanBackward0>)
```

[247]:
```
x0_int.size()
X0 = x0_int[0]
X0 = torch.reshape(X0, (X0.size(0), -1))
```

[248]:
```
X0.size()
```

[248]:
```
torch.Size([8, 1024])
```

[249]:
```
X0 = X0[:, 0:8]
X0.size()
```

[249]:
```
torch.Size([8, 8])
```

[250]:
```
bit_precision = 16
file = open('./resnet_out/residual.txt', 'w') #write to file
file.write('#time0row7[msb-lsb],time0row6[msb-lst],....,time0row0[msb-lst]#\n')
file.write('#time1row7[msb-lsb],time1row6[msb-lst],....,time1row0[msb-lst]#\n')
file.write('#................#\n')

for i in range(X0.size(1)):  # time step
    for j in range(X0.size(0)): # row #
        X0_bin = '{0:016b}'.format(int(X0[7-j,i].item()+0.001))
        for k in range(bit_precision):
            file.write(X0_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close() #close file
```

[251]:
```
X0
```

[251]:
```
tensor([[49., 35., 35., 35., 35., 35., 35., 56.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [49., 14., 14., 14., 14., 14., 14.,  7.],
        [28., 14., 14., 14., 14., 14., 14., 14.],
        [ 0.,  7.,  7.,  0.,  0.,  0.,  0.,  0.],
        [77., 14., 14., 14., 14., 14., 14., 21.],
        [42., 35., 35., 35., 35., 35., 35., 35.],
        [ 7., 21., 21., 21., 28., 28., 21., 28.]], device='cuda:0',
       grad_fn=<SliceBackward>)
```

[252]:
```
answer = out + X0
print(answer)
```

```
tensor([[-11.,  31.,  33.,  35.,  48.,  22.,  28.,  65.],
        [  8.,  37.,  46.,  50.,  38.,  42.,  35.,  21.],
        [ 30.,  16.,  19.,  19.,  21.,  21.,  -8.,  34.],
        [ 28.,   7.,   6.,   9.,  13.,  35.,  10.,  -2.],
        [ 19.,   9.,   5.,  -5.,  -4.,   5.,   1.,  20.],
        [109.,  -2.,  -1.,  -2.,  -7., -12.,  30., -53.],
        [ 21., -41., -35., -32., -20., -27., -35., -24.],
        [ 54.,  43.,  40.,  39.,  37.,  43.,  25.,  25.]], device='cuda:0',
       grad_fn=<AddBackward0>)
```

[253]:
```
bit_precision = 16
file = open('./resnet_out/answer.txt', 'w') #write to file
file.write('#time0col7[msb-lsb],time0col6[msb-lst],....,time0col0[msb-lst]#\n')
file.write('#time1col7[msb-lsb],time1col6[msb-lst],....,time1col0[msb-lst]#\n')
file.write('#...............#\n')

for i in range(answer.size(1)):
    for j in range(answer.size(0)):
        if (answer[7-j,i].item()<0):
            ans_bin = '{0:016b}'.format(int(answer[7-j,i].
 ↪item()+2**bit_precision+0.001))
        else:
            ans_bin = '{0:016b}'.format(int(answer[7-j,i].item()+0.001))
        for k in range(bit_precision):
            file.write(ans_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close()
```

[266]:
```
f = open('./resnet_out/address.txt','r')
line = f.readlines()
bit_precision =11
file = open('./resnet_out/acc_address.txt','w')
for item in line:
    add = '{0:011b}'.format(int(item))
    for k in range(bit_precision):
        file.write(add[k])
    file.write('\n')
file.close()
```

[ ]:
```

```