[HW6_Prob2]_Memory_Write_Read

November 18, 2021

[2]:
```
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

#from tensorboardX import SummaryWriter

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 = "VGG16_quant"
model = VGG16_quant()
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 = [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…
VGG_quant(
  (features): Sequential(
    (0): QuantConv2d(
      3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): QuantConv2d(
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): QuantConv2d(
      64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
```

```
    )
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): QuantConv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (14): QuantConv2d(
      128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): QuantConv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): QuantConv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): QuantConv2d(
      256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
    (29): ReLU(inplace=True)
    (30): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (34): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (36): ReLU(inplace=True)
    (37): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (42): ReLU(inplace=True)
    (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (44): AvgPool2d(kernel_size=1, stride=1, padding=0)
  )
  (classifier): Linear(in_features=512, out_features=10, bias=True)
)
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
# HW

#  1. Load your saved model and validate
#  2. Replace your model's all the Conv's weight with quantized weight
#  3. Apply reasonable alpha
```

```
#  4. Then, try to multiple bit precisions and draw graph of bit precision vs. accuracy
```

[3]:
```
PATH = "result/VGG16_quant/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)))
```

```
/opt/conda/lib/python3.9/site-packages/torch/nn/functional.py:718: UserWarning:
Named tensors and all their associated APIs are an experimental feature and
subject to change. Please do not use them for anything important until they are
released as stable. (Triggered internally at
/pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation,
ceil_mode)


Test set: Accuracy: 9050/10000 (90%)
```

[4]:
```
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()
i = 0

for layer in model.modules():
    i = i+1
    if isinstance(layer, QuantConv2d):
        print(i,"-th layer prehooked")
        layer.register_forward_pre_hook(save_output)
#####################################################

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

```
3 -th layer prehooked
7 -th layer prehooked
12 -th layer prehooked
16 -th layer prehooked
21 -th layer prehooked
25 -th layer prehooked
29 -th layer prehooked
34 -th layer prehooked
38 -th layer prehooked
42 -th layer prehooked
47 -th layer prehooked
51 -th layer prehooked
55 -th layer prehooked
```

[5]:
```
weight_q = model.features[3].weight_q
w_alpha = model.features[3].weight_quant.wgt_alpha
w_bit = 4

weight_int = weight_q / (w_alpha / (2**(w_bit-1)-1))
print(weight_int)
```

```
tensor([[[[ 1.0000, -1.0000, -4.0000],
          [ 3.0000,  4.0000,  1.0000],
          [-1.0000, -3.0000, -3.0000]],

         [[-3.0000, -3.0000, -3.0000],
          [-2.0000, -5.0000, -5.0000],
          [-0.0000, -0.0000, -3.0000]],

         [[-0.0000,  0.0000, -0.0000],
          [-0.0000, -0.0000, -1.0000],
          [ 0.0000, -0.0000, -0.0000]],
```

```
 …,

 [[-4.0000, -5.0000, -4.0000],
  [-3.0000, -4.0000, -4.0000],
  [-0.0000, -1.0000, -1.0000]],

 [[ 2.0000,  2.0000,  1.0000],
  [ 1.0000,  2.0000,  1.0000],
  [-0.0000, -0.0000, -2.0000]],

 [[-1.0000, -2.0000, -2.0000],
  [-4.0000, -3.0000, -1.0000],
  [-4.0000, -2.0000, -0.0000]]],


[[[ 0.0000,  5.0000,  2.0000],
  [-1.0000, -0.0000,  4.0000],
  [-1.0000, -5.0000, -3.0000]],

 [[-1.0000,  2.0000,  3.0000],
  [-1.0000, -1.0000,  3.0000],
  [-1.0000, -2.0000,  1.0000]],

 [[ 0.0000, -0.0000, -0.0000],
  [ 1.0000,  1.0000,  0.0000],
  [ 1.0000,  1.0000,  0.0000]],

 …,

 [[-1.0000,  1.0000,  4.0000],
  [-1.0000, -1.0000,  1.0000],
  [-1.0000, -1.0000, -0.0000]],

 [[ 0.0000, -0.0000, -2.0000],
  [ 1.0000, -0.0000, -1.0000],
  [ 1.0000,  0.0000, -1.0000]],

 [[-2.0000, -1.0000,  0.0000],
  [-5.0000, -4.0000, -2.0000],
  [-4.0000, -3.0000, -2.0000]]],


[[[ 2.0000,  7.0000,  2.0000],
  [ 6.0000, -2.0000, -7.0000],
  [-4.0000, -3.0000, -2.0000]],

 [[-3.0000, -2.0000,  2.0000],
  [ 3.0000,  1.0000,  0.0000],
```

```
  [ 2.0000, -2.0000, -3.0000]],

 [[-1.0000, -0.0000, -0.0000],
  [-0.0000,  0.0000, -0.0000],
  [-1.0000,  0.0000, -0.0000]],

 …,

 [[ 0.0000, -0.0000,  0.0000],
  [ 1.0000,  2.0000,  2.0000],
  [ 2.0000,  1.0000, -1.0000]],

 [[ 1.0000,  0.0000,  0.0000],
  [ 0.0000, -0.0000, -0.0000],
  [-0.0000, -0.0000, -0.0000]],

 [[-4.0000, -4.0000, -5.0000],
  [-7.0000, -5.0000, -6.0000],
  [-4.0000, -3.0000, -1.0000]]],


…,


[[[ 2.0000,  6.0000,  6.0000],
  [-4.0000, -3.0000,  1.0000],
  [-6.0000, -6.0000, -5.0000]],

 [[-1.0000, -0.0000,  1.0000],
  [-1.0000, -1.0000,  1.0000],
  [-1.0000, -1.0000,  0.0000]],

 [[ 2.0000,  1.0000,  1.0000],
  [ 2.0000,  2.0000,  1.0000],
  [ 1.0000,  1.0000,  1.0000]],

 …,

 [[-2.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000],
  [ 1.0000,  1.0000, -0.0000]],

 [[-0.0000,  0.0000,  1.0000],
  [-1.0000, -0.0000, -0.0000],
  [ 0.0000, -0.0000,  0.0000]],

 [[-1.0000, -0.0000,  1.0000],
  [-0.0000, -1.0000,  1.0000],
```

```
  [ 1.0000,  1.0000,  1.0000]]],


[[[ 1.0000, -5.0000, -7.0000],
  [ 7.0000, -0.0000, -7.0000],
  [ 7.0000,  4.0000, -4.0000]],

 [[ 1.0000,  3.0000,  3.0000],
  [ 1.0000,  3.0000,  2.0000],
  [ 2.0000,  2.0000,  2.0000]],

 [[ 1.0000,  1.0000,  1.0000],
  [-0.0000,  1.0000,  1.0000],
  [-1.0000, -0.0000,  1.0000]],

 …,

 [[ 3.0000,  3.0000,  2.0000],
  [ 3.0000,  4.0000,  3.0000],
  [-1.0000,  1.0000,  0.0000]],

 [[ 4.0000,  4.0000,  3.0000],
  [ 3.0000,  4.0000,  3.0000],
  [ 2.0000,  3.0000,  2.0000]],

 [[-3.0000, -1.0000, -0.0000],
  [ 0.0000,  2.0000,  1.0000],
  [ 1.0000,  3.0000,  2.0000]]],


[[[-5.0000,  2.0000,  7.0000],
  [ 0.0000, -1.0000, -2.0000],
  [ 4.0000,  4.0000, -4.0000]],

 [[ 3.0000, -1.0000, -1.0000],
  [ 1.0000, -2.0000, -0.0000],
  [-1.0000, -2.0000,  1.0000]],

 [[-2.0000, -2.0000, -2.0000],
  [-2.0000, -2.0000, -1.0000],
  [-2.0000, -1.0000, -1.0000]],

 …,

 [[-7.0000, -7.0000, -7.0000],
  [-5.0000, -3.0000,  1.0000],
  [ 3.0000,  7.0000,  7.0000]],
```

```
          [[ 0.0000, -2.0000, -2.0000],
           [-1.0000, -2.0000, -1.0000],
           [ 0.0000, -0.0000,  0.0000]],

          [[-0.0000, -2.0000, -2.0000],
           [-2.0000, -2.0000, -2.0000],
           [ 1.0000,  1.0000,  3.0000]]]], device='cuda:0',
       grad_fn=<DivBackward0>)
```

[6]:
```
act = save_output.outputs[1][0]
act_alpha  = model.features[3].act_alpha
act_bit = 4
act_quant_fn = act_quantization(act_bit)

act_q = act_quant_fn(act, act_alpha)

act_int = act_q / (act_alpha / (2**act_bit-1))
print(act_int)
```

```
tensor([[[[15.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
          [15.0000,  9.0000,  8.0000,  …,  4.0000,  2.0000,  0.0000],
          [15.0000,  9.0000,  7.0000,  …,  6.0000,  5.0000,  0.0000],
          …,
          [ 0.0000,  9.0000,  5.0000,  …, 13.0000, 10.0000, 15.0000],
          [ 0.0000,  7.0000,  6.0000,  …,  5.0000, 12.0000, 15.0000],
          [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  7.0000]],

         [[ 0.0000,  0.0000,  0.0000,  …,  1.0000,  1.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  …,  1.0000,  1.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  2.0000,  0.0000],
          …,
          [ 6.0000,  5.0000,  1.0000,  …,  7.0000,  1.0000,  0.0000],
          [ 6.0000,  3.0000,  1.0000,  …,  5.0000,  2.0000,  0.0000],
          [ 4.0000,  1.0000,  1.0000,  …,  2.0000,  1.0000,  0.0000]],

         [[ 1.0000,  2.0000,  2.0000,  …,  1.0000,  1.0000,  1.0000],
          [ 0.0000,  2.0000,  2.0000,  …,  1.0000,  0.0000,  1.0000],
          [ 0.0000,  1.0000,  2.0000,  …,  1.0000,  0.0000,  1.0000],
          …,
          [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

         …,

         [[ 1.0000,  1.0000,  1.0000,  …,  1.0000,  2.0000,  2.0000],
          [ 2.0000,  3.0000,  2.0000,  …,  4.0000,  4.0000,  3.0000],
          [ 2.0000,  2.0000,  3.0000,  …,  4.0000,  4.0000,  3.0000],
```

```
  …,
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  1.0000,  0.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  1.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

  [[ 4.0000,  5.0000,  5.0000,  …,  5.0000,  4.0000,  4.0000],
   [ 6.0000,  7.0000,  7.0000,  …,  6.0000,  6.0000,  5.0000],
   [ 6.0000,  8.0000,  8.0000,  …,  6.0000,  6.0000,  5.0000],
   …,
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

  [[ 1.0000,  2.0000,  3.0000,  …,  0.0000,  0.0000,  9.0000],
   [ 1.0000,  5.0000,  7.0000,  …,  2.0000,  2.0000, 14.0000],
   [ 0.0000,  4.0000,  7.0000,  …,  3.0000,  2.0000, 13.0000],
   …,
   [ 0.0000,  0.0000,  3.0000,  …,  0.0000,  4.0000, 15.0000],
   [ 0.0000,  0.0000,  3.0000,  …,  0.0000,  2.0000, 15.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  1.0000, 14.0000]]],


 [[[13.0000, 14.0000, 14.0000,  …, 14.0000, 15.0000, 15.0000],
   [ 6.0000,  1.0000,  2.0000,  …,  1.0000,  1.0000, 15.0000],
   [ 6.0000,  0.0000,  1.0000,  …,  1.0000,  1.0000, 15.0000],
   …,
   [ 8.0000,  9.0000,  7.0000,  …,  3.0000,  2.0000, 15.0000],
   [ 6.0000,  7.0000,  6.0000,  …,  2.0000,  6.0000, 15.0000],
   [ 0.0000,  0.0000,  0.0000,  …, 12.0000, 14.0000, 15.0000]],

  [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  9.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 12.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 12.0000],
   …,
   [11.0000, 12.0000,  8.0000,  …,  0.0000,  0.0000,  6.0000],
   [11.0000,  9.0000,  5.0000,  …,  0.0000,  0.0000,  6.0000],
   [ 7.0000,  6.0000,  3.0000,  …,  0.0000,  0.0000,  3.0000]],

  [[ 3.0000,  3.0000,  3.0000,  …,  3.0000,  3.0000,  2.0000],
   [ 4.0000,  5.0000,  5.0000,  …,  5.0000,  5.0000,  2.0000],
   [ 4.0000,  5.0000,  5.0000,  …,  5.0000,  5.0000,  2.0000],
   …,
   [ 0.0000,  0.0000,  0.0000,  …,  3.0000,  3.0000,  1.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  3.0000,  3.0000,  1.0000],
   [ 0.0000,  0.0000,  0.0000,  …,  3.0000,  3.0000,  1.0000]],

  …,
```

```
[[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
 [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
 [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
 …,
 [ 2.0000,  5.0000,  6.0000,  …,  0.0000,  0.0000,  0.0000],
 [ 2.0000,  4.0000,  5.0000,  …,  0.0000,  0.0000,  0.0000],
 [ 2.0000,  3.0000,  4.0000,  …,  0.0000,  0.0000,  0.0000]],

[[ 1.0000,  1.0000,  1.0000,  …,  1.0000,  1.0000,  0.0000],
 [ 2.0000,  1.0000,  1.0000,  …,  1.0000,  1.0000,  0.0000],
 [ 2.0000,  1.0000,  1.0000,  …,  1.0000,  1.0000,  0.0000],
 …,
 [ 0.0000,  0.0000,  0.0000,  …,  2.0000,  1.0000,  0.0000],
 [ 0.0000,  1.0000,  1.0000,  …,  1.0000,  1.0000,  0.0000],
 [ 1.0000,  1.0000,  2.0000,  …,  1.0000,  1.0000,  0.0000]],

[[15.0000,  8.0000,  9.0000,  …, 10.0000,  8.0000,  0.0000],
 [15.0000,  5.0000,  7.0000,  …,  7.0000,  6.0000,  0.0000],
 [15.0000,  5.0000,  6.0000,  …,  6.0000,  6.0000,  0.0000],
 …,
 [ 0.0000,  0.0000,  0.0000,  …, 14.0000, 11.0000,  0.0000],
 [ 0.0000,  0.0000,  0.0000,  …, 13.0000, 10.0000,  0.0000],
 [ 0.0000,  0.0000,  2.0000,  …, 10.0000,  8.0000,  0.0000]]],


[[[ 3.0000, 15.0000, 14.0000,  …, 14.0000, 15.0000, 15.0000],
  [ 0.0000,  8.0000,  5.0000,  …,  0.0000,  2.0000, 15.0000],
  [ 0.0000,  6.0000,  4.0000,  …,  0.0000,  1.0000, 13.0000],
  …,
  [ 0.0000, 10.0000,  9.0000,  …, 15.0000, 10.0000,  0.0000],
  [ 0.0000, 10.0000,  8.0000,  …, 13.0000,  9.0000,  2.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[ 0.0000,  0.0000,  2.0000,  …,  0.0000,  0.0000,  9.0000],
  [ 0.0000,  0.0000,  3.0000,  …,  0.0000,  0.0000, 13.0000],
  [ 0.0000,  0.0000,  4.0000,  …,  0.0000,  0.0000, 14.0000],
  …,
  [15.0000,  1.0000,  1.0000,  …,  5.0000,  6.0000,  0.0000],
  [15.0000,  1.0000,  0.0000,  …,  7.0000,  4.0000,  0.0000],
  [13.0000,  1.0000,  1.0000,  …,  7.0000,  3.0000,  0.0000]],

 [[ 2.0000,  1.0000,  1.0000,  …,  3.0000,  3.0000,  2.0000],
  [ 3.0000,  2.0000,  1.0000,  …,  5.0000,  5.0000,  2.0000],
  [ 3.0000,  2.0000,  2.0000,  …,  5.0000,  5.0000,  2.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],
```

```
 …,

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  …,
  [ 6.0000,  4.0000,  5.0000,  …,  5.0000,  6.0000,  5.0000],
  [ 7.0000,  5.0000,  5.0000,  …,  6.0000,  7.0000,  6.0000],
  [ 6.0000,  6.0000,  6.0000,  …,  9.0000,  9.0000,  6.0000]],

 [[ 0.0000,  0.0000,  0.0000,  …,  1.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  1.0000,  1.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  1.0000,  1.0000,  0.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  2.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  3.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  1.0000,  2.0000]],

 [[15.0000,  5.0000,  3.0000,  …, 10.0000, 11.0000,  0.0000],
  [15.0000,  0.0000,  0.0000,  …,  9.0000, 10.0000,  0.0000],
  [15.0000,  0.0000,  0.0000,  …,  9.0000, 10.0000,  0.0000],
  …,
  [ 0.0000,  3.0000,  4.0000,  …,  0.0000,  0.0000, 15.0000],
  [ 0.0000,  4.0000,  5.0000,  …,  0.0000,  0.0000, 15.0000],
  [ 0.0000,  6.0000,  7.0000,  …,  0.0000,  5.0000, 15.0000]]],


…,


[[[15.0000,  1.0000,  2.0000,  …,  0.0000,  0.0000,  0.0000],
  [15.0000,  2.0000,  1.0000,  …,  0.0000,  0.0000,  0.0000],
  [15.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  …,
  [15.0000, 10.0000,  6.0000,  …, 14.0000,  4.0000,  6.0000],
  [15.0000, 10.0000,  5.0000,  …,  0.0000,  2.0000,  7.0000],
  [15.0000, 15.0000, 15.0000,  …,  0.0000,  0.0000,  2.0000]],

 [[ 0.0000,  0.0000,  0.0000,  …,  9.0000,  7.0000,  0.0000],
  [ 0.0000,  2.0000,  0.0000,  …, 12.0000, 10.0000,  0.0000],
  [ 0.0000,  3.0000,  0.0000,  …, 11.0000,  9.0000,  0.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  2.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[ 2.0000,  2.0000,  2.0000,  …,  2.0000,  0.0000,  0.0000],
```

```
  [ 2.0000,  3.0000,  2.0000,  …,  2.0000,  0.0000,  0.0000],
  [ 2.0000,  3.0000,  2.0000,  …,  2.0000,  0.0000,  0.0000],
  …,
  [ 3.0000,  4.0000,  4.0000,  …,  1.0000,  0.0000,  0.0000],
  [ 2.0000,  4.0000,  3.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 1.0000,  2.0000,  2.0000,  …,  0.0000,  0.0000,  0.0000]],

 …,

 [[ 0.0000,  0.0000,  0.0000,  …,  4.0000,  4.0000,  3.0000],
  [ 0.0000,  1.0000,  1.0000,  …,  7.0000,  7.0000,  3.0000],
  [ 0.0000,  2.0000,  2.0000,  …,  7.0000,  6.0000,  3.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  2.0000,  0.0000,  1.0000],
  [ 1.0000,  2.0000,  1.0000,  …,  5.0000,  2.0000,  1.0000]],

 [[ 4.0000,  4.0000,  3.0000,  …,  4.0000,  2.0000,  2.0000],
  [ 5.0000,  5.0000,  5.0000,  …,  5.0000,  2.0000,  2.0000],
  [ 5.0000,  5.0000,  4.0000,  …,  5.0000,  2.0000,  2.0000],
  …,
  [ 7.0000,  8.0000,  7.0000,  …,  1.0000,  1.0000,  1.0000],
  [ 6.0000,  7.0000,  6.0000,  …,  2.0000,  1.0000,  1.0000],
  [ 4.0000,  5.0000,  4.0000,  …,  2.0000,  1.0000,  1.0000]],

 [[ 9.0000,  1.0000,  3.0000,  …,  0.0000,  0.0000, 10.0000],
  [12.0000,  0.0000,  2.0000,  …,  0.0000,  0.0000, 14.0000],
  [11.0000,  0.0000,  3.0000,  …,  0.0000,  0.0000, 14.0000],
  …,
  [15.0000, 10.0000,  0.0000,  …,  0.0000,  0.0000, 10.0000],
  [15.0000, 12.0000,  1.0000,  …, 14.0000,  6.0000, 10.0000],
  [12.0000, 13.0000,  6.0000,  …, 15.0000, 15.0000, 11.0000]]],


[[[ 4.0000, 13.0000, 13.0000,  …, 15.0000, 13.0000, 15.0000],
  [ 0.0000,  3.0000,  3.0000,  …,  5.0000,  4.0000, 15.0000],
  [ 0.0000,  3.0000,  3.0000,  …,  5.0000,  5.0000, 15.0000],
  …,
  [ 0.0000,  6.0000,  7.0000,  …, 13.0000,  7.0000,  9.0000],
  [ 0.0000,  6.0000,  8.0000,  …, 10.0000,  8.0000, 13.0000],
  [ 0.0000,  6.0000,  6.0000,  …,  0.0000,  0.0000,  2.0000]],

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  2.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  3.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  4.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
```

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

 [[ 1.0000,  1.0000,  1.0000,  …,  1.0000,  1.0000,  1.0000],
  [ 2.0000,  2.0000,  2.0000,  …,  2.0000,  2.0000,  1.0000],
  [ 2.0000,  2.0000,  2.0000,  …,  2.0000,  2.0000,  1.0000],
  …,
  [ 1.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 1.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 1.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  0.0000]],

 …,

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  2.0000,  3.0000,  2.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  2.0000,  2.0000,  2.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  2.0000,  2.0000,  2.0000]],

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  …,
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[15.0000,  8.0000,  8.0000,  …,  8.0000,  7.0000,  0.0000],
  [15.0000,  6.0000,  6.0000,  …,  5.0000,  4.0000,  0.0000],
  [15.0000,  6.0000,  6.0000,  …,  6.0000,  5.0000,  0.0000],
  …,
  [10.0000,  6.0000,  6.0000,  …,  5.0000,  5.0000, 12.0000],
  [ 9.0000,  6.0000,  6.0000,  …,  6.0000,  7.0000, 13.0000],
  [ 8.0000,  4.0000,  4.0000,  …,  7.0000,  7.0000, 12.0000]]],


[[[ 2.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [10.0000, 10.0000,  7.0000,  …, 10.0000, 11.0000,  2.0000],
  [10.0000,  8.0000,  8.0000,  …,  8.0000, 11.0000,  2.0000],
  …,
  [13.0000,  4.0000,  3.0000,  …, 11.0000, 11.0000,  2.0000],
  [ 8.0000,  2.0000, 10.0000,  …,  8.0000, 11.0000,  2.0000],
  [14.0000, 15.0000,  8.0000,  …,  0.0000,  0.0000,  0.0000]],

 [[ 9.0000,  1.0000,  1.0000,  …,  5.0000,  3.0000,  0.0000],
  [10.0000,  2.0000,  0.0000,  …,  5.0000,  3.0000,  0.0000],
  [11.0000,  2.0000,  0.0000,  …,  7.0000,  3.0000,  0.0000],
```

```
          …,
           [ 0.0000,  0.0000,  0.0000,  …,  6.0000,  3.0000,  0.0000],
           [ 0.0000,  0.0000,  0.0000,  …,  5.0000,  2.0000,  0.0000],
           [ 0.0000,  0.0000,  0.0000,  …,  5.0000,  3.0000,  0.0000]],

          [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
           [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
           [ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
           …,
           [ 2.0000,  2.0000,  2.0000,  …,  0.0000,  0.0000,  0.0000],
           [ 1.0000,  2.0000,  2.0000,  …,  0.0000,  0.0000,  0.0000],
           [ 1.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  0.0000]],

          …,

          [[ 4.0000,  3.0000,  4.0000,  …,  6.0000,  6.0000,  5.0000],
           [ 5.0000,  5.0000,  4.0000,  …,  7.0000,  8.0000,  6.0000],
           [ 5.0000,  5.0000,  4.0000,  …,  8.0000,  8.0000,  6.0000],
           …,
           [ 0.0000,  0.0000,  0.0000,  …,  7.0000,  7.0000,  6.0000],
           [ 0.0000,  0.0000,  0.0000,  …,  8.0000,  7.0000,  6.0000],
           [ 0.0000,  0.0000,  1.0000,  …, 10.0000, 10.0000,  7.0000]],

          [[ 1.0000,  2.0000,  2.0000,  …,  1.0000,  1.0000,  2.0000],
           [ 0.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  3.0000],
           [ 0.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  3.0000],
           …,
           [ 2.0000,  2.0000,  2.0000,  …,  0.0000,  0.0000,  3.0000],
           [ 2.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  3.0000],
           [ 1.0000,  1.0000,  1.0000,  …,  1.0000,  1.0000,  2.0000]],

          [[ 0.0000,  2.0000,  3.0000,  …,  0.0000,  2.0000, 15.0000],
           [ 0.0000,  3.0000,  7.0000,  …,  0.0000,  3.0000, 15.0000],
           [ 0.0000,  2.0000,  8.0000,  …,  0.0000,  2.0000, 15.0000],
           …,
           [15.0000,  6.0000,  5.0000,  …,  0.0000,  2.0000, 15.0000],
           [12.0000,  7.0000,  7.0000,  …,  0.0000,  4.0000, 15.0000],
           [11.0000,  5.0000,  5.0000,  …,  2.0000,  6.0000, 15.0000]]]],
       device='cuda:0', grad_fn=<DivBackward0>)
```

[7]:
```
## This cell is provided

conv_int = torch.nn.Conv2d(in_channels = 64, out_channels=64, kernel_size = 3,␣
 ↪padding=1)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
conv_int.bias = model.features[3].bias
output_int = conv_int(act_int)
```

```
output_recovered = output_int * (act_alpha / (2**act_bit-1)) * (w_alpha /␣
↪(2**(w_bit-1)-1))
print(output_recovered)
```

```
tensor([[[[-2.5817e+01, -3.8373e+01, -4.2938e+01,  …, -3.4351e+01,
           -2.8263e+01, -1.7556e+01],
          [-1.4893e+01, -1.9893e+01, -2.4295e+01,  …, -2.3480e+01,
           -1.8099e+01, -1.3153e+01],
          [-2.0600e+01, -2.8589e+01, -3.1253e+01,  …, -3.1144e+01,
           -2.5274e+01, -1.7991e+01],
          …,
          [-3.8644e+01, -5.2885e+01, -5.8048e+01,  …, -5.1852e+01,
           -5.2504e+01, -3.5927e+01],
          [-4.8482e+01, -6.7234e+01, -6.8756e+01,  …, -7.1310e+01,
           -7.0278e+01, -4.3917e+01],
          [-2.5111e+01, -3.2611e+01, -3.5057e+01,  …, -3.9786e+01,
           -3.9732e+01, -2.1850e+01]],

         [[-1.7284e+01, -1.1631e+01, -6.1962e+00,  …, -4.1308e+00,
           -5.0004e+00,  4.6743e+00],
          [-3.2394e+01, -4.4949e+01, -4.4351e+01,  …, -3.5927e+01,
           -3.7720e+01, -2.2067e+01],
          [-1.9186e+01, -3.1633e+01, -2.8698e+01,  …, -2.2665e+01,
           -2.5600e+01, -1.6632e+01],
          …,
          [ 2.1197e+00, -9.4029e+00, -9.0225e+00,  …, -1.0327e+00,
           -3.0546e+01, -2.0110e+01],
          [ 1.8480e+00,  2.9350e+00, -3.6416e+00,  …,  1.4023e+01,
           -9.0768e+00, -1.4349e+01],
          [-1.4838e+01, -4.5112e+00, -9.7834e+00,  …,  2.1741e-01,
            1.8480e+00, -1.4132e+00]],

         [[-2.2556e+01, -1.6414e+01, -2.6035e+01,  …, -1.2338e+01,
           -3.0981e+00, -2.7176e+00],
          [-2.2230e+01, -2.1795e+01, -3.1361e+01,  …, -1.3262e+01,
           -2.1469e+01, -2.2882e+01],
          [-1.7175e+01, -1.7121e+01, -2.6252e+01,  …, -2.2121e+01,
           -2.2665e+01, -1.6414e+01],
          …,
          [-1.5762e+01, -6.1418e+00, -2.8807e+00,  …,  1.0979e+01,
           -2.0328e+01, -2.5437e+01],
          [-8.2615e+00, -3.8590e+00, -1.7012e+01,  …,  2.1850e+01,
           -2.3263e+01, -1.0762e+01],
          [-9.5117e+00, -1.5327e+01, -2.0436e+01,  …, -1.2936e+01,
           -3.3807e+01, -2.1034e+01]],

         …,
```

```
 [[ 8.4246e+00,  2.8426e+01,  4.2232e+01,  …,  4.0656e+01,
    4.6308e+01,  4.3319e+01],
  [-1.3425e+01, -1.0599e+01,  2.1741e-01,  …, -1.1360e+01,
   -3.7503e+00,  1.5762e+01],
  [-1.4132e+00,  3.3155e+00,  1.4349e+01,  …, -3.1524e+00,
    1.6306e+00,  1.6197e+01],
  …,
  [ 1.1360e+01,  1.8915e+01,  1.4729e+01,  …,  6.1418e+00,
    9.1312e+00,  3.4242e+00],
  [ 2.7774e+01,  4.3536e+01,  3.4025e+01,  …,  3.9732e+01,
    3.3970e+01,  1.4240e+01],
  [ 1.6088e+01,  4.0112e+01,  3.7992e+01,  …,  4.5439e+01,
    4.1797e+01,  3.2611e+01]],

 [[ 1.3805e+01,  2.4024e+01,  5.1635e+00,  …,  9.4030e+00,
    1.4512e+01,  5.2178e+00],
  [ 3.1850e+01,  6.8647e+01,  5.2233e+01,  …,  5.5820e+01,
    5.7831e+01,  3.2231e+01],
  [ 2.4622e+01,  5.9027e+01,  4.4678e+01,  …,  5.4298e+01,
    5.8211e+01,  3.1470e+01],
  …,
  [-5.4352e+00, -3.7231e+01, -1.7447e+01,  …, -1.9404e+01,
   -1.7067e+01,  9.8921e+00],
  [-1.6632e+01, -5.5385e+01, -3.3481e+01,  …, -3.4079e+01,
   -4.4515e+01, -3.9134e+00],
  [ 4.6199e+00, -1.9512e+01, -8.6964e+00,  …, -8.8594e+00,
   -2.0980e+01, -5.0004e+00]],

 [[ 4.1743e+01,  6.4788e+01,  6.0277e+01,  …,  4.8102e+01,
    4.6308e+01,  2.5600e+01],
  [ 2.9350e+01,  2.6252e+01,  2.7774e+01,  …,  2.0273e+01,
    2.0491e+01,  6.0331e+00],
  [ 2.7665e+01,  2.3752e+01,  2.1143e+01,  …,  4.5656e+00,
    1.0870e+01,  5.2722e+00],
  …,
  [-1.7338e+01, -5.0004e+00,  9.1855e+00,  …, -2.3915e+00,
    4.9461e+00,  1.9676e+01],
  [-5.2722e+00,  1.9295e+01,  2.8644e+01,  …,  1.9132e+01,
    2.5219e+01,  3.6253e+01],
  [ 3.6470e+01,  5.7668e+01,  4.7286e+01,  …,  4.6960e+01,
    4.6580e+01,  2.1958e+01]]],


[[[-3.3861e+01, -5.0493e+01, -4.0492e+01,  …, -4.0112e+01,
   -5.5820e+01, -3.0981e+01],
  [-3.9732e+01, -6.1690e+01, -5.0874e+01,  …, -5.3211e+01,
   -7.0332e+01, -3.0655e+01],
```

```
 [-3.6144e+01, -5.9461e+01, -4.7232e+01,  …, -4.7069e+01,
  -7.0332e+01, -3.0166e+01],
 …,
 [-4.5167e+01, -6.1364e+01, -6.7451e+01,  …, -4.7993e+01,
  -6.5223e+01, -2.7339e+01],
 [-4.3971e+01, -5.6635e+01, -5.8646e+01,  …, -4.9787e+01,
  -6.2831e+01, -2.7122e+01],
 [-2.0600e+01, -2.5002e+01, -2.7067e+01,  …, -9.5660e+00,
  -1.6904e+01,  2.0654e+00]],

[[ 1.3425e+01,  7.3376e+00,  8.6420e+00,  …,  8.8594e+00,
   1.4621e+01,  3.5329e+00],
 [-1.0381e+01, -1.8860e+01, -9.7291e+00,  …, -1.4240e+01,
  -2.1741e-01,  3.5872e+00],
 [-1.4675e+01, -2.5274e+01, -1.5708e+01,  …, -1.7067e+01,
  -6.1418e+00,  1.0327e+00],
 …,
 [ 4.8916e-01,  4.7286e+00,  2.7176e+00,  …, -2.2991e+01,
  -1.6360e+01, -1.4132e+00],
 [ 8.9681e+00,  7.0114e+00,  7.6637e+00,  …, -2.1197e+01,
  -9.7291e+00,  4.7286e+00],
 [ 1.0925e+01,  9.6747e+00,  7.5006e+00,  …, -3.4133e+01,
  -2.0763e+01,  4.1308e+00]],

[[-3.3753e+01, -4.1634e+01, -2.8644e+01,  …, -3.3916e+01,
  -3.8699e+01, -2.0436e+01],
 [-3.0818e+01, -1.9404e+01, -6.5766e+00,  …, -6.5223e+00,
  -1.6958e+01,  1.3045e+00],
 [-2.1741e+01, -2.1415e+01, -1.6360e+01,  …, -1.7501e+01,
  -2.8481e+01, -5.6526e+00],
 …,
 [ 4.8918e-01,  2.0328e+01,  1.0871e-01,  …, -5.2233e+01,
  -4.7286e+01, -1.3479e+01],
 [ 9.3486e+00,  7.7724e+00, -1.4893e+01,  …, -5.2939e+01,
  -4.4732e+01, -3.3698e+00],
 [ 8.2072e+00, -1.3099e+01, -2.5872e+01,  …, -4.2123e+01,
  -2.3480e+01,  8.9138e+00]],

…,

[[ 2.2067e+01,  2.4948e+01,  1.8588e+01,  …,  2.3589e+01,
   1.2066e+01,  7.1201e+00],
 [ 2.2339e+01,  5.5331e+01,  4.6852e+01,  …,  4.6471e+01,
   4.0275e+01,  3.5166e+01],
 [ 1.1251e+01,  4.1797e+01,  3.3101e+01,  …,  3.0709e+01,
   2.6470e+01,  2.8861e+01],
 …,
 [ 3.5873e+00,  5.7613e+00,  4.2395e+00,  …,  1.5871e+01,
```

```
    1.4186e+01,  1.7230e+01],
   [ 7.3919e+00,  7.9898e+00,  7.3376e+00,  …,  2.8154e+01,
     2.6361e+01,  2.0871e+01],
   [ 1.1631e+01,  1.3425e+01,  8.9138e+00,  …,  3.4568e+01,
     4.7558e+01,  4.2667e+01]],

  [[ 1.0218e+01,  4.8917e+00,  8.0985e+00,  …,  5.0548e+00,
     4.5112e+00,  1.6577e+01],
   [ 1.7556e+01, -1.6306e-01, -1.3045e+00,  …, -2.8807e+00,
    -7.2832e+00,  1.0979e+01],
   [ 2.5383e+01,  8.8051e+00,  3.8047e+00,  …,  5.3265e+00,
    -4.0764e+00,  1.0001e+01],
   …,
   [ 1.3697e+01,  3.2557e+01,  4.2884e+01,  …,  2.0110e+00,
    -1.2610e+01,  2.6089e+00],
   [ 1.4729e+01,  3.1959e+01,  4.1145e+01,  …, -9.2399e+00,
    -2.0654e+01, -3.3698e+00],
   [ 1.8208e+01,  2.8481e+01,  3.7014e+01,  …,  7.6093e-01,
    -7.3919e+00,  7.0658e-01]],

  [[ 2.7177e-01,  1.4675e+00,  2.0654e+00,  …, -2.1741e+00,
     3.8047e-01,  4.0221e+00],
   [ 3.3698e+00, -1.2284e+01, -1.5599e+01,  …, -7.8811e+00,
    -7.2832e+00, -1.4838e+01],
   [-1.1958e+00, -1.6795e+01, -1.2229e+01,  …, -7.4463e+00,
    -2.7720e+00, -8.5333e+00],
   …,
   [-1.6849e+00, -5.4352e-01,  1.7936e+00,  …,  2.1741e-01,
     5.4896e+00,  3.1524e+00],
   [-2.2828e+00,  4.5656e+00,  4.1851e+00,  …,  1.1686e+01,
     1.3914e+01,  5.9244e+00],
   [ 1.2718e+01,  1.1794e+01,  8.5877e+00,  …,  1.4947e+01,
     1.9621e+01,  1.4675e+00]]],


 [[[-3.6090e+01, -5.3863e+01, -3.7068e+01,  …, -4.2177e+01,
    -6.2559e+01, -3.4514e+01],
   [-4.4841e+01, -6.9734e+01, -5.2993e+01,  …, -5.3537e+01,
    -7.5115e+01, -3.2557e+01],
   [-4.0058e+01, -6.0766e+01, -4.1906e+01,  …, -4.7178e+01,
    -7.6800e+01, -3.2068e+01],
   …,
   [-3.9460e+01, -5.2341e+01, -4.5982e+01,  …, -5.0602e+01,
    -4.5384e+01, -3.7503e+01],
   [-4.6852e+01, -6.0385e+01, -5.7722e+01,  …, -5.7233e+01,
    -5.9679e+01, -4.8808e+01],
   [-3.6362e+01, -4.4134e+01, -3.9786e+01,  …, -4.4025e+01,
    -4.8319e+01, -3.2394e+01]],
```

```
[[ 1.9893e+01,  1.0436e+01, -9.2399e-01,  …,  9.0768e+00,
   1.3588e+01,  2.6089e+00],
 [ 1.9023e+00,  3.2068e+00, -4.1851e+00,  …, -1.5762e+01,
  -4.5112e+00,  3.2611e-01],
 [-1.2012e+01, -1.1142e+01, -1.3045e+01,  …, -2.2719e+01,
  -1.2012e+01, -3.2611e+00],
 …,
 [-1.4729e+01, -1.7664e+01, -1.6795e+01,  …, -1.4675e+01,
  -2.4676e+01, -2.0328e+01],
 [-1.1794e+01, -1.2066e+01, -1.2392e+01,  …, -6.8484e+00,
  -2.2284e+01, -2.5056e+01],
 [-2.6633e+00, -4.5112e+00, -6.7397e+00,  …,  3.9134e+00,
  -1.0544e+01, -1.9839e+01]],

[[-2.6796e+01, -4.0819e+01, -1.7991e+01,  …, -2.6959e+01,
  -4.3536e+01, -2.5980e+01],
 [-3.7394e+01, -2.7122e+01, -4.8918e-01,  …, -7.5550e+00,
  -2.8915e+01, -2.3371e+00],
 [-2.8861e+01, -2.0165e+01, -2.6089e+00,  …, -1.9458e+01,
  -3.5818e+01, -5.5439e+00],
 …,
 [-1.4240e+01, -4.2938e+00, -5.9787e+00,  …,  2.9894e+00,
   3.0763e+01,  1.6306e-01],
 [-1.1034e+01, -1.0381e+01, -1.8480e+01,  …,  3.2666e+01,
   1.1360e+01, -2.1795e+01],
 [-1.1794e+01, -2.4404e+01, -2.8752e+01,  …,  5.9244e+00,
  -3.0329e+01, -4.5167e+01]],

…,

[[ 1.5817e+01,  6.2505e+00, -1.2501e+00,  …,  2.5654e+01,
   1.6306e+01,  1.0381e+01],
 [ 2.9676e+01,  5.0276e+01,  3.4242e+01,  …,  5.1200e+01,
   4.6091e+01,  4.0547e+01],
 [ 1.4566e+01,  3.7775e+01,  2.4024e+01,  …,  2.9024e+01,
   2.9459e+01,  3.4568e+01],
 …,
 [ 9.1312e+00,  9.6204e+00,  7.7724e+00,  …,  1.3479e+01,
   2.1795e+01,  2.0436e+01],
 [ 1.0436e+01,  1.5708e+01,  2.2339e+01,  …,  2.5491e+01,
   2.7339e+01,  1.6469e+01],
 [ 5.9788e+00,  6.7940e+00,  1.4947e+01,  …,  1.7882e+01,
   8.4790e+00,  6.8484e+00]],

[[-1.4675e+00, -1.4512e+01,  7.5006e+00,  …,  5.0004e+00,
   3.0437e+00,  1.3969e+01],
 [-9.4029e+00, -4.3917e+01, -1.7393e+01,  …,  4.1851e+00,
```

```
   -4.7286e+00,  1.0490e+01],
  [ 2.1741e+00, -2.3698e+01,  3.3155e+00,  …,  2.0926e+01,
   -2.1741e-01,  9.2399e+00],
  …,
  [ 2.1252e+01,  2.3371e+00,  1.3099e+01,  …,  4.4025e+00,
    2.5872e+01,  3.5655e+01],
  [ 2.5600e+01,  7.2832e+00,  1.3479e+01,  …,  1.8371e+01,
    3.6144e+01,  4.2721e+01],
  [ 2.9948e+01,  2.2611e+01,  2.1958e+01,  …,  3.4188e+01,
    4.2667e+01,  3.4568e+01]],

 [[-2.9187e+01, -2.7883e+01, -1.3697e+01,  …,  1.1251e+01,
    1.0707e+01,  7.1201e+00],
  [-8.5333e+00,  1.0870e-01,  5.7070e+00,  …,  6.3592e+00,
   -4.0764e+00, -1.8317e+01],
  [-6.5223e-01,  4.1308e+00,  3.2611e+00,  …, -1.5219e+00,
   -7.4463e+00, -1.6686e+01],
  …,
  [ 9.6204e+00,  2.3861e+01,  1.5762e+01,  …,  2.1415e+01,
    1.9839e+01,  9.9465e+00],
  [ 6.5766e+00,  2.0708e+01,  8.8594e+00,  …,  1.5871e+01,
    1.0925e+01,  2.2284e+00],
  [ 1.1414e+01,  1.5708e+01,  6.8484e+00,  …, -1.3045e+00,
   -1.3860e+01, -2.0708e+01]]],


…,


[[[-3.1633e+01, -4.1580e+01, -3.2666e+01,  …, -4.3101e+01,
   -2.8644e+01, -1.5653e+01],
  [-1.8915e+01, -2.5980e+01, -1.8643e+01,  …, -4.2395e+01,
   -2.9731e+01, -1.8371e+01],
  [-2.9839e+01, -3.5764e+01, -2.4839e+01,  …, -4.6254e+01,
   -3.5492e+01, -2.2882e+01],
  …,
  [-1.6632e+01, -3.5655e+01, -3.5166e+01,  …, -2.3589e+01,
   -3.3481e+01, -2.0763e+01],
  [-1.3371e+01, -2.9024e+01, -2.9350e+01,  …, -4.0275e+01,
   -3.4133e+01, -2.3480e+01],
  [-4.4569e+00, -7.7724e+00, -3.3155e+00,  …, -3.2340e+01,
   -2.6415e+01, -1.2664e+01]],

 [[-8.9138e+00, -6.7397e+00,  3.0981e+00,  …, -3.0981e+00,
   -1.1631e+01, -4.4569e+00],
  [-3.1524e+01, -4.7178e+01, -3.4188e+01,  …, -2.4839e+01,
   -3.2448e+01, -2.2774e+01],
  [-1.7447e+01, -3.1633e+01, -1.9241e+01,  …, -1.4512e+01,
```

```
  -3.0437e+01, -2.3589e+01],
 …,
 [-3.4079e+01, -3.9623e+01, -1.7447e+01,  …, -1.7284e+01,
  -1.7121e+01, -7.2832e+00],
 [-3.7884e+01, -5.1852e+01, -3.2720e+01,  …, -4.6797e+01,
  -3.0926e+01, -1.5273e+01],
 [-1.7991e+01, -3.8047e+01, -2.9568e+01,  …, -4.5982e+01,
  -3.6090e+01, -1.8262e+01]],

[[-3.6525e+01, -3.2503e+01, -2.3589e+01,  …,  2.1197e+00,
    1.9241e+01,  1.1740e+01],
 [-2.5763e+01, -7.8811e+00,  4.8917e+00,  …,  3.2557e+01,
    1.4023e+01, -8.1528e+00],
 [-8.1528e+00, -5.1091e+00, -3.1524e+00,  …,  2.2828e+01,
    1.0327e+01, -9.6204e+00],
 …,
 [-2.5491e+01, -1.4838e+01,  2.4459e+00,  …, -9.2399e-01,
    3.8046e-01, -6.2505e+00],
 [-2.1904e+01, -1.5871e+01, -8.1528e-01,  …, -1.9730e+01,
  -3.4731e+01, -2.7774e+01],
 [-1.4186e+01, -1.6632e+01, -3.2611e-01,  …, -5.6581e+01,
  -5.5602e+01, -3.6742e+01]],

…,

[[ 1.2338e+01,  3.1796e+01,  4.1036e+01,  …,  2.4350e+01,
    3.9786e+01,  3.1144e+01],
 [-1.6088e+01,  1.5219e+00,  6.8484e+00,  …, -7.6093e+00,
    4.4569e+00,  1.4784e+01],
 [-1.1142e+01,  3.0981e+00,  1.2827e+01,  …,  1.0870e+00,
    1.0490e+01,  1.5273e+01],
 …,
 [ 3.5873e+00,  1.6469e+01,  2.1089e+01,  …,  5.3320e+01,
    2.4785e+01,  1.3860e+01],
 [-3.8047e+00,  1.6306e+00,  6.5223e+00,  …,  4.1199e+01,
    5.0222e+01,  2.5437e+01],
 [-3.0981e+00, -1.0327e+00,  1.0327e+00,  …, -1.5490e+01,
    2.9350e+00,  2.0817e+01]],

[[ 1.4132e+01,  1.6034e+01, -4.3482e-01,  …,  3.4242e+01,
    1.1088e+01, -7.6093e-01],
 [ 3.9134e+01,  6.0929e+01,  4.0601e+01,  …,  9.3269e+01,
    5.0167e+01,  1.1414e+01],
 [ 2.9731e+01,  5.2722e+01,  2.4187e+01,  …,  7.7343e+01,
    4.2286e+01,  9.0768e+00],
 …,
 [ 2.5437e+01,  4.7232e+01,  4.0547e+01,  …,  1.0599e+01,
    5.4351e-02,  3.6416e+00],
```

```
  [ 2.2176e+01,  4.7069e+01,  4.4623e+01,  …,  2.3535e+01,
    2.9894e+00,  3.3698e+00],
  [ 8.5333e+00,  2.5056e+01,  3.0546e+01,  …,  3.2122e+01,
    1.2936e+01,  2.7176e-01]],

 [[ 4.5710e+01,  6.1744e+01,  4.9243e+01,  …,  7.1310e+01,
    2.7502e+01,  6.7940e+00],
  [ 3.0600e+01,  1.3969e+01,  5.5983e+00,  …, -1.0979e+01,
   -2.4187e+01, -8.6420e+00],
  [ 1.5273e+01, -5.6526e+00, -6.0331e+00,  …, -3.2068e+01,
   -3.2883e+01, -8.3703e+00],
  …,
  [ 7.0658e+00, -1.4077e+01, -1.4023e+01,  …,  6.9571e+00,
    2.5437e+01,  2.7230e+01],
  [ 6.6310e+00, -1.2881e+01, -9.8378e+00,  …,  1.1088e+01,
    4.2938e+00,  1.4349e+01],
  [-1.3208e+01, -3.4079e+01, -2.9676e+01,  …, -3.3155e+00,
    2.8807e+00,  5.2178e+00]]],


[[[-2.6687e+01, -4.2558e+01, -3.7449e+01,  …, -3.8808e+01,
   -4.5493e+01, -2.5709e+01],
  [-3.5329e+01, -5.7287e+01, -5.1961e+01,  …, -5.1743e+01,
   -5.8755e+01, -2.6252e+01],
  [-3.0926e+01, -5.0222e+01, -4.4243e+01,  …, -4.3427e+01,
   -5.3320e+01, -2.3154e+01],
  …,
  [-2.3698e+01, -3.9623e+01, -3.8808e+01,  …, -3.7394e+01,
   -3.8590e+01, -2.8318e+01],
  [-3.0818e+01, -4.9298e+01, -4.7939e+01,  …, -4.8102e+01,
   -4.4678e+01, -3.0111e+01],
  [-9.8921e+00, -1.7012e+01, -1.5436e+01,  …, -2.7937e+01,
   -3.0546e+01, -1.7556e+01]],

 [[ 1.3099e+01,  7.7180e+00,  7.8811e+00,  …,  9.2942e+00,
    8.6420e+00,  1.7936e+00],
  [-8.0985e+00, -1.1686e+01, -5.1091e+00,  …, -1.0327e+00,
    3.5329e+00,  7.3919e+00],
  [-1.4295e+01, -2.1034e+01, -1.5436e+01,  …, -1.4675e+01,
   -7.3376e+00,  2.5546e+00],
  …,
  [-1.4729e+01, -2.2176e+01, -2.0871e+01,  …, -2.1850e+01,
   -2.6307e+01, -2.0110e+01],
  [-8.5877e+00, -1.2338e+01, -1.2012e+01,  …, -1.4023e+01,
   -1.9404e+01, -1.6360e+01],
  [-1.9784e+01, -1.7121e+01, -1.3697e+01,  …, -1.2936e+01,
   -1.8588e+01, -1.1631e+01]],
```

```
[[-2.5383e+01, -3.5655e+01, -2.2991e+01,  …, -2.2176e+01,
  -2.6361e+01, -1.2338e+01],
 [-3.1361e+01, -2.7611e+01, -1.3371e+01,  …, -1.5762e+01,
  -1.7501e+01,  4.8917e-01],
 [-2.7230e+01, -3.1035e+01, -2.1360e+01,  …, -2.1306e+01,
  -2.3643e+01, -2.1197e+00],
 …,
 [-2.0926e+01, -2.4948e+01, -2.0165e+01,  …, -1.0870e+01,
  -1.0925e+01, -1.6849e+01],
 [-2.4893e+01, -3.2122e+01, -2.5600e+01,  …, -9.0225e+00,
  -3.0057e+01, -2.2502e+01],
 [-2.6959e+01, -2.6741e+01, -1.8480e+01,  …, -2.6470e+01,
  -3.2611e+01, -2.1741e+01]],

…,

[[ 1.9839e+01,  1.7012e+01,  6.5223e+00,  …,  5.1091e+00,
  -5.6526e+00, -8.2072e+00],
 [ 2.3915e+01,  4.5602e+01,  3.7720e+01,  …,  3.9242e+01,
   3.0437e+01,  1.8697e+01],
 [ 1.1034e+01,  2.8644e+01,  1.8317e+01,  …,  1.8262e+01,
   1.6577e+01,  1.2338e+01],
 …,
 [ 5.7613e+00,  1.5382e+01,  6.4679e+00,  …,  4.7830e+00,
   4.4025e+00,  7.1745e+00],
 [ 1.5980e+01,  2.6904e+01,  1.7828e+01,  …,  2.6415e+01,
   2.8263e+01,  1.9349e+01],
 [ 1.1142e+01,  3.5438e+01,  3.2883e+01,  …,  1.6034e+01,
   1.6904e+01,  1.9078e+01]],

[[ 2.7720e+00, -9.6747e+00,  2.1741e+00,  …, -3.2611e+00,
  -2.5002e+00,  1.0327e+01],
 [ 6.5223e-01, -3.0818e+01, -1.9784e+01,  …, -2.5491e+01,
  -2.8426e+01, -3.9134e+00],
 [ 9.4573e+00, -1.7501e+01, -4.5656e+00,  …, -7.0658e+00,
  -1.7556e+01,  2.3371e+00],
 …,
 [ 5.0004e+00, -1.6577e+01,  4.3481e-01,  …,  2.1741e-01,
   3.4785e+00,  1.1142e+01],
 [-7.2289e+00, -3.4188e+01, -1.7828e+01,  …, -1.2284e+01,
  -9.5117e+00,  3.7503e+00],
 [ 2.5546e+00, -1.7610e+01, -9.4573e+00,  …,  1.4675e+00,
   5.4352e-02,  1.7393e+00]],

[[-2.0763e+01, -2.6687e+01, -2.1578e+01,  …, -2.3861e+01,
  -2.4078e+01, -4.3482e+00],
 [-5.9787e+00, -3.5329e+00, -3.2611e-01,  …,  3.0437e+00,
  -1.0327e+00,  6.7940e+00],
```

```
  [-3.8590e+00, -1.8480e+00,  1.0870e+00,  …,  4.6743e+00,
    2.9350e+00,  7.6093e+00],
  …,
  [-4.5112e+00,  1.1958e+00,  7.2832e+00,  …, -5.4352e-01,
    7.0658e+00,  1.3805e+01],
  [-4.1851e+00,  1.1088e+01,  1.2284e+01,  …,  1.3914e+01,
    1.3153e+01,  1.5871e+01],
  [ 2.3263e+01,  3.3970e+01,  2.5437e+01,  …,  1.6469e+01,
    1.1468e+01,  1.9567e+00]]],


[[[-2.8100e+01, -3.4785e+01, -4.0384e+01,  …, -4.4895e+01,
   -4.1036e+01, -2.9296e+01],
  [-2.8807e+01, -3.2611e+01, -3.4622e+01,  …, -5.3537e+01,
   -5.2233e+01, -4.0329e+01],
  [-3.4894e+01, -4.0873e+01, -4.1090e+01,  …, -6.0820e+01,
   -5.5494e+01, -4.1906e+01],
  …,
  [-2.3752e+01, -4.3047e+01, -4.6036e+01,  …, -5.7179e+01,
   -5.6200e+01, -4.2395e+01],
  [-2.8915e+01, -3.7286e+01, -2.5546e+01,  …, -6.5821e+01,
   -6.6799e+01, -5.0819e+01],
  [-3.5329e+00, -1.0436e+01, -1.1958e+01,  …, -5.1472e+01,
   -5.1961e+01, -3.3427e+01]],

 [[-1.6795e+01, -1.3697e+01, -1.1903e+01,  …, -1.7447e+01,
   -2.8589e+01, -1.3153e+01],
  [-1.5490e+01, -2.1360e+01, -2.0002e+01,  …, -1.5708e+01,
   -3.2340e+01, -2.8915e+01],
  [-8.0985e+00, -2.1578e+01, -2.0002e+01,  …, -9.2942e+00,
   -2.2393e+01, -2.2230e+01],
  …,
  [-2.7448e+01, -3.9732e+01, -3.5927e+01,  …, -4.0221e+00,
   -2.0436e+01, -2.2176e+01],
  [-2.7394e+01, -4.4786e+01, -3.2013e+01,  …, -1.0544e+01,
   -2.3100e+01, -2.5817e+01],
  [-1.3099e+01, -3.1905e+01, -3.3590e+01,  …, -1.7936e+00,
   -1.2447e+01, -2.0871e+01]],

 [[-3.0981e+00,  1.0218e+01,  3.0437e+00,  …,  1.1577e+01,
    7.6093e+00,  3.8047e+00],
  [-1.5164e+01, -1.0001e+01, -3.1959e+01,  …, -1.4132e+01,
   -1.4349e+01, -2.2882e+01],
  [-9.2942e+00, -3.4786e+00, -2.3045e+01,  …,  4.8917e+00,
    1.3045e+00, -1.2773e+01],
  …,
  [-1.7121e+01,  1.4675e+00,  3.2068e+00,  …,  6.3592e+00,
    8.1528e-01, -1.3425e+01],
```

```
 [-1.5871e+01, -3.5981e+01, -4.0601e+01,  …,  7.4463e+00,
  -9.4029e+00, -1.9295e+01],
 [-2.4785e+01, -1.5708e+01,  8.5333e+00,  …, -1.3588e+01,
  -2.8154e+01, -4.0221e+01]],

…,

[[ 8.6964e-01,  1.0381e+01,  1.6958e+01,  …,  2.4839e+01,
   3.5492e+01,  3.4188e+01],
 [-8.8051e+00, -1.5436e+01, -7.1745e+00,  …, -6.3049e+00,
  -1.0327e+00,  6.6310e+00],
 [ 8.1528e+00,  9.5660e+00,  1.5056e+01,  …,  1.2447e+01,
   2.1252e+01,  1.5762e+01],
 …,
 [ 6.0875e+00,  2.6307e+01,  1.4893e+01,  …,  2.0219e+01,
   2.2339e+01,  1.5980e+01],
 [-9.4573e+00,  1.0001e+01,  8.7507e+00,  …,  2.5926e+01,
   3.1090e+01,  2.1034e+01],
 [-3.0981e+00,  2.9894e+00,  2.5002e+00,  …,  9.8921e+00,
   7.4463e+00,  8.3703e+00]],

[[ 2.0817e+01,  1.9621e+01,  1.5001e+01,  …,  2.3426e+01,
   2.2013e+01,  2.5219e+01],
 [ 2.0600e+01,  2.5274e+01,  1.8752e+01,  …,  2.1958e+01,
   2.8263e+01,  3.7557e+01],
 [ 1.4566e+01,  2.1089e+01,  1.4893e+01,  …,  2.6633e+01,
   3.3264e+01,  4.2503e+01],
 …,
 [ 8.0441e+00,  7.7724e+00,  9.2399e-01,  …,  2.0056e+01,
   3.1850e+01,  4.2069e+01],
 [ 1.2555e+01,  1.3914e+01,  1.1142e+01,  …,  3.0709e+01,
   3.4949e+01,  4.3210e+01],
 [ 8.9138e+00,  8.9138e+00,  9.8921e+00,  …,  4.3591e+01,
   4.3645e+01,  3.4949e+01]],

[[ 8.0441e+00,  1.5273e+01,  1.1468e+01,  …,  1.3045e+00,
   1.2990e+01,  1.4947e+01],
 [ 1.0490e+01,  2.2284e+01,  2.3208e+01,  …,  1.0762e+01,
   1.8480e+01,  7.9898e+00],
 [ 1.4675e+01,  2.5926e+01,  2.9024e+01,  …,  8.7507e+00,
   9.1312e+00, -9.2399e-01],
 …,
 [ 1.7610e+01,  7.3376e+00,  7.2289e+00,  …,  6.3049e+00,
   1.1740e+01, -3.8047e-01],
 [ 1.3099e+01,  5.4896e+00,  2.2828e+00,  …,  9.5660e+00,
   6.7397e+00, -5.9277e-07],
 [-5.4353e-02, -1.3534e+01, -8.8051e+00,  …, -4.3482e+00,
  -1.4512e+01, -2.2447e+01]]]], device='cuda:0',
```

```
        grad_fn=<MulBackward0>)
```

[8]:
```
## This cell is provided

conv_ref = torch.nn.Conv2d(in_channels = 64, out_channels=64, kernel_size = 3,␣
 ↪padding=1)
conv_ref.weight = model.features[3].weight_q
conv_ref.bias = model.features[3].bias
output_ref = conv_ref(act)
#print(output_ref)

print(abs((output_ref - output_recovered)).mean())
```

```
tensor(2.3094, device='cuda:0', grad_fn=<MeanBackward0>)
```

[9]:
```
# act_int.size = torch.Size([128, 64, 32, 32])  <- batch_size, input_ch, ni, nj
a_int = act_int[0,:,:,:]  # pick only one input out of batch
# a_int.size() = [64, 32, 32]

# conv_int.weight.size() = torch.Size([64, 64, 3, 3])  <- output_ch, input_ch,␣
 ↪ki, kj
w_int = torch.reshape(weight_int, (weight_int.size(0), weight_int.size(1), -1))␣
 ↪ # merge ki, kj index to kij
# w_int.weight.size() = torch.Size([64, 64, 9])

padding = 1
stride = 1
array_size = 8 # row and column number

nig = range(a_int.size(1))  ## ni group
njg = range(a_int.size(2))  ## nj group

icg = range(int(w_int.size(1)))  ## input channel
ocg = range(int(w_int.size(0)))  ## output channel

ic_tileg = range(int(len(icg)/array_size))
oc_tileg = range(int(len(ocg)/array_size))

kijg = range(w_int.size(2))
ki_dim = int(math.sqrt(w_int.size(2)))  ## Kernel's 1 dim size

######## Padding before Convolution #######
a_pad = torch.zeros(len(icg), len(nig)+padding*2, len(nig)+padding*2).cuda()
# a_pad.size() = [64, 32+2pad, 32+2pad]
a_pad[ :, padding:padding+len(nig), padding:padding+len(njg)] = a_int.cuda()
a_pad = torch.reshape(a_pad, (a_pad.size(0), -1))
# a_pad.size() = [64, (32+2pad)*(32+2pad)]
```

```
a_tile = torch.zeros(len(ic_tileg), array_size,    a_pad.size(1)).cuda()
w_tile = torch.zeros(len(oc_tileg)*len(ic_tileg), array_size, array_size,␣
↪len(kijg)).cuda()

for ic_tile in ic_tileg:
    a_tile[ic_tile,:,:] = a_pad[ic_tile*array_size:(ic_tile+1)*array_size,:]

for ic_tile in ic_tileg:
    for oc_tile in oc_tileg:
        w_tile[oc_tile*len(oc_tileg) + ic_tile,:,:,:] =␣
↪w_int[oc_tile*array_size:(oc_tile+1)*array_size, ic_tile*array_size:
↪(ic_tile+1)*array_size, :]



##########################################

p_nijg = range(a_pad.size(1)) ## psum nij group

psum = torch.zeros(len(ic_tileg), len(oc_tileg), array_size, len(p_nijg),␣
↪len(kijg)).cuda()

for kij in kijg:
    for ic_tile in ic_tileg:       # Tiling into array_sizeXarray_size array
        for oc_tile in oc_tileg:   # Tiling into array_sizeXarray_size array   ␣
↪
            for nij in p_nijg:       # time domain, sequentially given input
                    m = nn.Linear(array_size, array_size, bias=False)
                    #m.weight = torch.nn.Parameter(w_int[oc_tile*array_size:
↪(oc_tile+1)*array_size, ic_tile*array_size:(ic_tile+1)*array_size, kij])
                    m.weight = torch.nn.
↪Parameter(w_tile[len(oc_tileg)*oc_tile+ic_tile,:,:,kij])
                    psum[ic_tile, oc_tile, :, nij, kij] = m(a_tile[ic_tile,:
↪,nij]).cuda()
```

[10]:
```
import math

a_pad_ni_dim = int(math.sqrt(a_pad.size(1))) # 32

o_ni_dim = int((a_pad_ni_dim - (ki_dim- 1) - 1)/stride + 1)
o_nijg = range(o_ni_dim**2)

out = torch.zeros(len(ocg), len(o_nijg)).cuda()
```

```
### SFP accumulation ###
for o_nij in o_nijg:
    for kij in kijg:
        for ic_tile in ic_tileg:
            for oc_tile in oc_tileg:
                out[oc_tile*array_size:(oc_tile+1)*array_size, o_nij] =␣
↪out[oc_tile*array_size:(oc_tile+1)*array_size, o_nij] + \
                psum[ic_tile, oc_tile, :, int(o_nij/o_ni_dim)*a_pad_ni_dim +␣
↪o_nij%o_ni_dim + int(kij/ki_dim)*a_pad_ni_dim + kij%ki_dim, kij]
                ## 4th index = (int(o_nij/30)*32 + o_nij%30) + (int(kij/3)*32 +␣
↪kij%3)
```

[11]:
```
out_2D = torch.reshape(out, (out.size(0), o_ni_dim, -1))
difference = (out_2D - output_int[0,:,:,:])
print(difference.sum())
```

```
tensor(0.0653, device='cuda:0', grad_fn=<SumBackward0>)
```

[12]:
```
### show this cell partially. The following cells should be printed by students␣
↪###
tile_id = 0
nij = 200 # just a random number
X = a_tile[tile_id,:,nij:nij+64]  # [tile_num, array row num, time_steps]

bit_precision = 4
file = open('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
```

[13]:
```
X[:,0] # check this number with your first line in activation.txt
```

[13]:
```
tensor([13.,  0.,  0.,  0.,  0.,  0.,  0.,  2.], device='cuda:0',
       grad_fn=<SelectBackward>)
```

[14]:
```
### Complete this cell ###
tile_id = 0
kij = 0
W = w_tile[tile_id,:,:,kij]  # w_tile[tile_num, array col num, array row num,␣
↪kij]


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 i in range(W.size(0)): # time step
    for j in range(W.size(1)): # row #
        if (W[7-j,i].item() > 0):
            W_bin = '{0:04b}'.format(int(W[7-j,i].item()+0.001))
        else:
            W_bin = '{0:04b}'.format(int(W[7-j,i].item() + 0.001)+16)
        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
```

[15]:
```
W[0,:] # check this number with your 2nd line in weight.txt
```

[15]:
```
tensor([ 1.0000, -3.0000, -0.0000, -0.0000, -1.0000, -3.0000,  0.0000, -1.0000],
       device='cuda:0', grad_fn=<SliceBackward>)
```

[22]:
```
### Complete this cell ###
ic_tile_id = 0
oc_tile_id = 0


kij = 0
nij = 200
psum_tile = psum[ic_tile_id,oc_tile_id,:,nij:nij+64,kij]
# psum[len(ic_tileg), len(oc_tileg), array_size, len(p_nijg), len(kijg)]


bit_precision = 16
file = open('psum.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(psum_tile.size(1)): # time step
    for j in range(psum_tile.size(0)): # row #
        if (psum_tile[7-j,i].item()>0):
            psum_tile_bin = '{0:016b}'.format(int(psum_tile[7-j,i].item()+0.
 ↪001))
        else:
            psum_tile_bin = '{0:016b}'.format(int(psum_tile[7-j,i].item()+0.
 ↪001)+65536)
        for k in range(bit_precision):
            file.write(psum_tile_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close() #close file
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: