[HW5_prob1]_VGGNet_Hardware_Mapping_with_Tiling

November 1, 2021

[1]:

```
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 = "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
```

```python
        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 from HW3_Prob2
#  2. Like W4S2_example, map your input and output onto 2D systolic array
#  3. But, this time, our array size is 16 X 16 unlike W4S2_example (64 X 64).␣
 ↪Thus, tiling is required.
```

[2]:

```
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: 9259/10000 (93%)
```

[3]:

```
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
```

[4]:

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

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

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

         …,

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

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

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


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

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

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

         …,

         [[-7.0000, -3.0000,  4.0000],
          [-2.0000,  7.0000, -5.0000],
          [ 2.0000, -4.0000,  1.0000]],

         [[ 1.0000,  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]]],


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

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

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

         …,

         [[ 6.0000, -0.0000, -2.0000],
          [ 2.0000, -7.0000, -2.0000],
          [-6.0000, -2.0000,  7.0000]],

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

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


        …,


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

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

         …,

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

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

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

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

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

         …,

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

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

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


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

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

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

         …,

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

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

[5]:
```
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([[[[ 9.0000,  7.0000,  7.0000,  …,  6.0000,  5.0000,  2.0000],
          [12.0000,  8.0000,  5.0000,  …,  6.0000,  5.0000,  1.0000],
          [12.0000,  8.0000,  6.0000,  …,  5.0000,  5.0000,  2.0000],
          …,
          [ 0.0000,  5.0000,  2.0000,  …,  6.0000,  7.0000,  5.0000],
          [ 0.0000,  5.0000,  3.0000,  …,  6.0000,  4.0000, 10.0000],
          [ 4.0000,  6.0000,  6.0000,  …,  6.0000,  3.0000,  7.0000]],

         [[ 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]],

         [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000, 13.0000],
          [ 0.0000,  2.0000,  4.0000,  …,  5.0000,  7.0000, 15.0000],
          [ 0.0000,  1.0000,  3.0000,  …,  6.0000,  6.0000, 15.0000],
          …,
          [15.0000,  3.0000,  5.0000,  …,  0.0000,  1.0000,  0.0000],
          [15.0000,  1.0000,  3.0000,  …,  4.0000,  0.0000,  0.0000],
          [12.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000]],


         …,

         [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  9.0000],
          [ 1.0000,  6.0000,  5.0000,  …,  6.0000,  8.0000, 15.0000],
          [ 1.0000,  5.0000,  4.0000,  …,  6.0000,  7.0000, 15.0000],
          …,
          [14.0000, 11.0000,  9.0000,  …,  8.0000, 15.0000,  1.0000],
          [14.0000,  7.0000,  9.0000,  …, 13.0000,  6.0000,  9.0000],
  [ 9.0000,  0.0000,  2.0000,  …,  4.0000,  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],
  …,
  [ 0.0000,  1.0000,  1.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 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,  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,  8.0000,  8.0000,  …,  8.0000,  8.0000,  8.0000],
  [ 0.0000,  8.0000,  7.0000,  …,  7.0000,  7.0000,  4.0000],
  [ 0.0000,  7.0000,  7.0000,  …,  7.0000,  7.0000,  3.0000],
  …,
  [ 6.0000, 10.0000,  7.0000,  …,  3.0000,  2.0000,  4.0000],
  [ 8.0000,  9.0000,  4.0000,  …,  3.0000,  5.0000,  5.0000],
  [10.0000,  8.0000,  3.0000,  …,  4.0000,  5.0000, 11.0000]],

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

 [[10.0000, 10.0000,  9.0000,  …,  9.0000, 10.0000,  0.0000],
  [10.0000,  9.0000,  9.0000,  …,  9.0000, 10.0000,  0.0000],
  [10.0000,  9.0000, 10.0000,  …, 10.0000, 10.0000,  0.0000],
  …,
  [ 1.0000,  0.0000,  0.0000,  …, 11.0000,  9.0000,  0.0000],
  [ 2.0000,  0.0000,  1.0000,  …, 10.0000,  8.0000,  0.0000],
  [ 2.0000,  1.0000,  4.0000,  …,  9.0000,  7.0000,  0.0000]],

 …,

 [[ 0.0000, 11.0000, 10.0000,  …,  9.0000, 10.0000,  0.0000],
  [ 0.0000,  6.0000,  4.0000,  …,  5.0000,  5.0000,  0.0000],
  [ 0.0000,  4.0000,  5.0000,  …,  5.0000,  5.0000,  0.0000],
  …,
  [ 7.0000, 13.0000, 12.0000,  …,  7.0000,  4.0000,  0.0000],
  [ 9.0000, 13.0000,  9.0000,  …,  5.0000,  5.0000,  0.0000],
  [11.0000, 11.0000,  9.0000,  …,  4.0000,  3.0000,  0.0000]],

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

 [[ 0.0000,  0.0000,  0.0000,  …,  0.0000,  0.0000,  0.0000],
  [ 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,  …,  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,  5.0000,  6.0000,  …,  8.0000,  7.0000,  7.0000],
  [ 0.0000,  5.0000,  8.0000,  …,  8.0000,  4.0000,  4.0000],
  [ 0.0000,  5.0000,  8.0000,  …,  9.0000,  5.0000,  5.0000],
  …,
  [13.0000,  3.0000,  3.0000,  …,  2.0000,  8.0000,  8.0000],
  [14.0000,  1.0000,  2.0000,  …,  6.0000,  6.0000,  7.0000],
  [15.0000,  0.0000,  1.0000,  …,  4.0000,  0.0000,  0.0000]],

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

 [[15.0000, 10.0000, 11.0000,  …, 11.0000, 10.0000,  0.0000],
  [15.0000,  4.0000,  4.0000,  …, 10.0000,  9.0000,  0.0000],
  [15.0000,  4.0000,  3.0000,  …, 11.0000,  8.0000,  0.0000],
  …,
  [ 8.0000,  3.0000,  4.0000,  …,  0.0000,  0.0000, 10.0000],
  [ 9.0000,  5.0000,  4.0000,  …,  0.0000,  1.0000, 11.0000],
  [ 4.0000,  2.0000,  2.0000,  …,  0.0000,  2.0000, 12.0000]],

 …,
 [[ 5.0000, 12.0000, 15.0000,  …, 13.0000, 11.0000,  0.0000],
  [ 0.0000,  3.0000,  7.0000,  …,  7.0000,  2.0000,  0.0000],
  [ 0.0000,  2.0000,  6.0000,  …,  9.0000,  1.0000,  0.0000],
  …,
  [15.0000,  7.0000,  9.0000,  …,  2.0000, 11.0000, 15.0000],
  [15.0000,  7.0000,  9.0000,  …,  8.0000, 10.0000, 15.0000],
  [15.0000,  4.0000,  5.0000,  …,  9.0000,  6.0000, 13.0000]],

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

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


…,


[[[ 4.0000,  8.0000,  6.0000,  …, 11.0000,  3.0000,  4.0000],
  [ 2.0000,  9.0000,  6.0000,  …,  6.0000,  0.0000,  6.0000],
  [ 3.0000,  6.0000,  5.0000,  …,  5.0000,  3.0000,  6.0000],
  …,
  [ 3.0000,  7.0000,  5.0000,  …, 11.0000, 14.0000,  5.0000],
  [ 5.0000,  6.0000,  4.0000,  …,  0.0000,  2.0000,  6.0000],
  [ 4.0000,  4.0000,  2.0000,  …,  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,  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,  3.0000,  2.0000,  …,  2.0000, 15.0000, 15.0000],
  [ 0.0000, 10.0000, 10.0000,  …, 15.0000, 15.0000, 15.0000],
  [ 0.0000, 12.0000, 10.0000,  …, 15.0000, 15.0000, 15.0000],
  …,
  [ 0.0000,  8.0000,  9.0000,  …,  6.0000,  5.0000,  5.0000],
  [ 0.0000,  9.0000,  8.0000,  …, 13.0000,  9.0000,  6.0000],
  [ 3.0000, 14.0000, 14.0000,  …, 10.0000,  9.0000,  6.0000]],

 …,

 [[ 0.0000,  3.0000,  0.0000,  …,  3.0000,  6.0000, 14.0000],
  [ 0.0000, 10.0000,  8.0000,  …, 15.0000, 15.0000, 15.0000],
  [ 0.0000, 10.0000,  7.0000,  …, 14.0000, 15.0000, 15.0000],
  …,
  [ 0.0000,  6.0000,  4.0000,  …,  7.0000, 15.0000,  7.0000],
  [ 0.0000,  4.0000,  2.0000,  …,  0.0000,  6.0000,  9.0000],
  [ 3.0000, 10.0000,  6.0000,  …,  4.0000,  0.0000,  6.0000]],

 [[ 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]],

 [[ 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]]],


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

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

 [[15.0000, 11.0000, 11.0000,  …, 10.0000, 12.0000,  0.0000],
  [15.0000,  8.0000,  8.0000,  …,  7.0000,  8.0000,  0.0000],
  [15.0000,  8.0000,  8.0000,  …,  8.0000,  8.0000,  0.0000],
  …,
  [14.0000,  7.0000,  5.0000,  …,  5.0000,  7.0000,  5.0000],
  [15.0000,  7.0000,  6.0000,  …,  4.0000,  7.0000,  4.0000],
  [ 9.0000,  2.0000,  3.0000,  …,  1.0000,  2.0000,  3.0000]],

 …,

 [[ 6.0000, 14.0000, 14.0000,  …, 13.0000, 13.0000,  4.0000],
  [ 1.0000,  7.0000,  7.0000,  …,  6.0000,  6.0000,  0.0000],
  [ 1.0000,  7.0000,  7.0000,  …,  6.0000,  6.0000,  0.0000],
  …,
  [ 7.0000,  8.0000,  6.0000,  …,  7.0000, 10.0000,  8.0000],
  [ 8.0000,  7.0000,  6.0000,  …,  7.0000,  8.0000,  6.0000],
  [ 3.0000,  4.0000,  4.0000,  …,  2.0000,  4.0000,  4.0000]],

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


[[[11.0000,  2.0000,  5.0000,  …,  2.0000,  1.0000,  1.0000],
  [11.0000,  4.0000,  5.0000,  …,  2.0000,  3.0000,  6.0000],
  [11.0000,  6.0000,  4.0000,  …,  2.0000,  2.0000,  7.0000],
  …,
  [ 0.0000,  8.0000,  7.0000,  …,  5.0000,  2.0000,  7.0000],
  [ 2.0000,  3.0000,  3.0000,  …,  5.0000,  2.0000,  7.0000],
  [ 1.0000,  8.0000,  7.0000,  …,  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,  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,  4.0000,  …,  5.0000,  5.0000, 15.0000],
          [ 2.0000,  5.0000,  5.0000,  …,  4.0000,  2.0000, 11.0000],
          [ 1.0000,  4.0000,  3.0000,  …,  4.0000,  2.0000, 11.0000],
          …,
          [ 5.0000,  7.0000,  7.0000,  …,  4.0000,  2.0000, 12.0000],
          [ 4.0000,  8.0000,  8.0000,  …,  3.0000,  2.0000, 12.0000],
          [ 6.0000,  5.0000,  6.0000,  …,  2.0000,  2.0000, 12.0000]],

         …,

         [[13.0000,  1.0000,  5.0000,  …,  5.0000,  5.0000, 14.0000],
          [15.0000,  9.0000,  8.0000,  …, 10.0000,  9.0000, 15.0000],
          [13.0000,  9.0000,  6.0000,  …, 10.0000,  8.0000, 15.0000],
          …,
          [ 0.0000,  6.0000,  5.0000,  …, 12.0000,  8.0000, 15.0000],
          [ 0.0000,  4.0000,  2.0000,  …, 11.0000,  8.0000, 15.0000],
          [ 1.0000,  6.0000,  9.0000,  …,  8.0000,  6.0000, 13.0000]],

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

[6]:

```
## 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([[[[-4.4458e+00,  3.7653e+00,  4.9448e+00,  …,  1.0253e+01,
            1.1432e+01,  1.3564e+01],
          [-1.5288e+01, -1.7375e+01, -1.6876e+01,  …, -2.9397e+01,
           -3.0349e+01, -1.2657e+01],
          [-1.4517e+00, -2.7673e+00,  0.0000e+00,  …, -4.4912e+00,
           -2.1322e+00,  5.4892e+00],
          …,
          [ 9.7535e+00,  7.1677e+00,  1.0752e+01,  …, -1.5424e+01,
           -1.6740e+01, -2.0414e+00],
          [ 1.7239e+01,  1.3292e+01,  7.0770e+00,  …,  1.6740e+01,
            4.2643e+00, -4.2643e+00],
          [ 1.8736e+01,  2.3908e+01,  1.7511e+01,  …,  2.5631e+01,
            2.5087e+01,  9.7989e+00]],

         [[ 1.2430e+01,  3.9468e+00,  4.9902e+00,  …,  2.3590e+00,
            1.9053e+00,  9.0277e+00],
          [ 8.9823e+00,  1.2203e+01,  8.1657e+00,  …,  8.4833e+00,
            1.1296e+01, -8.8009e+00],
          [ 1.2203e+01,  9.1638e+00, -6.1697e+00,  …, -5.3985e+00,
           -6.5326e+00, -6.3058e+00],
          …,
          [-2.1322e+01, -7.8482e+00,  1.8146e+00,  …, -1.1568e+01,
            1.9190e+01, -2.8081e+01],
          [-9.8896e+00, -7.2584e-01, -1.1704e+01,  …,  3.0622e+01,
           -1.4925e+01,  2.2683e-01],
          [-4.3097e+00, -1.7692e+00,  5.3985e+00,  …,  1.0797e+01,
           -1.0343e+01,  2.7083e+01]],

         [[ 1.2249e+01,  1.1296e+01,  2.0868e+00,  …,  2.8126e+00,
            1.4063e+00, -1.6558e+01],
          [ 9.4813e+00, -2.2683e-01, -6.3511e-01,  …, -7.7575e+00,
           -5.6253e+00, -1.8418e+01],
          [ 6.8502e+00,  6.3512e-01, -2.2683e+00,  …, -4.2643e+00,
           -7.3492e+00, -1.9371e+01],
          …,
          [-2.1367e+01, -2.4452e+01, -5.8068e+00,  …, -2.3862e+01,
           -6.6687e+00,  1.6921e+01],
          [-3.0304e+01, -1.7965e+01, -6.7594e+00,  …, -3.4251e+01,
           -5.0809e+00,  1.3610e-01],
          [-1.7602e+01, -5.7614e+00, -2.6312e+00,  …, -3.6292e-01,
           -2.4497e+00,  1.7375e+01]],


         …,


         [[ 2.8127e+00,  4.8087e+00,  1.0434e+00,  …,  8.2111e+00,
            1.1114e+01,  8.4833e+00],
          [ 8.8462e+00,  9.9350e+00,  5.6253e+00,  …,  6.5780e+00,
            7.6214e+00,  8.3018e+00],
  [ 8.1658e+00,  7.3492e+00,  8.1204e+00,  …,  1.1387e+01,
    1.3746e+01,  1.6059e+01],
  …,
  [ 2.2773e+01,  3.6020e+01,  3.2708e+01,  …,  3.8923e+01,
    2.7174e+01,  2.9669e+01],
  [ 2.6085e+01,  3.2391e+01,  3.3117e+01,  …,  3.3389e+01,
    3.5430e+01,  2.8489e+01],
  [ 1.9008e+01,  2.1957e+01,  2.0550e+01,  …,  1.8554e+01,
    2.9624e+01,  1.2249e+01]],

 [[-1.7148e+01, -2.4180e+01, -3.1121e+01,  …, -2.8172e+01,
   -3.5067e+01, -2.7310e+01],
  [-2.1594e+01, -2.7491e+01, -3.6202e+01,  …, -2.5042e+01,
   -3.0984e+01, -2.8308e+01],
  [-2.4497e+01, -2.8081e+01, -3.5839e+01,  …, -2.2547e+01,
   -2.8081e+01, -2.5268e+01],
  …,
  [-2.9805e+01, -4.3732e+01, -5.1989e+01,  …, -4.8904e+01,
   -5.5709e+01, -4.1645e+01],
  [-2.7537e+01, -4.2870e+01, -4.7271e+01,  …, -4.6590e+01,
   -5.6933e+01, -4.4277e+01],
  [-1.0933e+01, -2.0369e+01, -2.0777e+01,  …, -2.4815e+01,
   -2.6902e+01, -2.8172e+01]],

 [[-3.8560e+00, -2.0414e+00, -2.6766e+00,  …, -7.1677e+00,
   -9.2545e+00, -6.3511e+00],
  [-1.4608e+01, -1.8055e+01, -1.4290e+01,  …, -2.0142e+01,
   -9.8896e+00, -5.6253e+00],
  [-5.4438e+00, -3.5385e+00, -5.6707e+00,  …, -1.5243e+01,
   -9.2092e+00, -7.7575e+00],
  …,
  [-2.8127e+00, -1.3519e+01, -9.2999e+00,  …, -5.2170e+00,
   -1.2430e+01, -9.5267e+00],
  [ 1.4063e+01,  1.5470e+01,  1.9552e+01,  …,  2.6176e+01,
    2.4497e+01,  8.8916e+00],
  [ 5.4892e+00, -2.9487e+00, -9.6174e+00,  …, -1.1024e+01,
   -1.3020e+01, -1.4335e+01]]],


[[[-3.5793e+01, -6.3421e+01, -5.7387e+01,  …, -5.7251e+01,
   -5.9655e+01, -3.7472e+01],
  [ 4.1645e+01,  4.7407e+01,  5.7523e+01,  …,  5.5255e+01,
    4.8813e+01,  3.1801e+01],
  [ 1.2249e+01, -1.6331e+00,  1.1341e+00,  …,  7.7121e-01,
   -4.5368e-02,  6.2150e+00],
  …,
  [ 1.3882e+01,  1.1795e+01,  4.0828e-01,  …, -7.8028e+00,
   -4.6273e+00,  1.0888e+00],
 [-4.5365e-01, -4.5819e+00, -8.3018e+00,  …,  5.1263e+00,
   5.0355e+00,  8.3926e+00],
 [ 4.9902e+00,  2.6312e+00,  1.9507e+00,  …,  3.2663e+00,
   2.4044e+00,  1.7692e+00]],

[[-9.5267e-01, -2.8671e+01, -1.1341e+01,  …, -1.3247e+01,
  -7.3492e+00, -2.6448e+01],
 [-3.6701e+01, -7.2585e-01, -3.9921e+00,  …, -5.8521e+00,
  -4.1533e-06,  6.8955e+00],
 [-3.8560e+00,  4.8541e+00, -5.1716e+00,  …, -3.4478e+00,
  -6.9409e+00, -8.3018e+00],
 …,
 [-1.8191e+01,  2.8943e+01,  8.6194e+00,  …, -9.5268e-01,
  -2.6357e+01, -7.0316e+00],
 [ 3.6746e+00,  1.8645e+01, -2.1412e+01,  …, -1.9008e+01,
  -5.6707e+00,  3.6292e+00],
 [ 7.1677e+00, -1.7239e+00, -1.1387e+01,  …, -9.9350e+00,
  -4.4458e+00, -7.9843e+00]],

[[ 1.6332e+01,  1.1341e+00,  1.9507e+00,  …,  6.5780e+00,
   4.3097e+00,  7.9389e+00],
 [ 2.0868e+00, -4.8994e+00,  2.1322e+00,  …,  5.4438e-01,
  -1.3156e+00, -3.4931e+00],
 [ 8.8916e+00,  5.3077e+00,  4.4458e+00,  …,  4.5365e+00,
   4.6726e+00, -4.0375e+00],
 …,
 [-1.3065e+01, -4.1600e+01, -1.2385e+01,  …,  1.2022e+01,
   1.2748e+01, -2.0414e+00],
 [-1.9552e+01, -2.3953e+01, -1.5288e+01,  …,  1.5243e+01,
   1.4562e+01,  5.3077e+00],
 [-1.6513e+01, -5.2170e+00, -1.0888e+00,  …, -7.2584e-01,
  -5.4438e-01, -2.2864e+01]],


…,


[[ 3.8560e+00,  1.6332e+01,  9.2545e+00,  …,  8.3926e+00,
   2.0868e+00,  1.1568e+01],
 [ 2.1821e+01,  2.8489e+01,  2.0505e+01,  …,  2.0823e+01,
   1.3428e+01,  2.3182e+01],
 [ 2.2592e+01,  2.8716e+01,  1.7829e+01,  …,  1.7783e+01,
   1.1704e+01,  2.2138e+01],
 …,
 [ 2.7038e+01,  2.7038e+01,  1.4199e+01,  …,  1.3610e+01,
   1.5923e+01,  2.2138e+01],
 [ 2.2637e+01,  1.3292e+01,  1.7193e+01,  …,  2.5994e+01,
   2.6720e+01,  2.6675e+01],
 [ 1.6921e+01,  1.6377e+01,  2.4044e+01,  …,  1.7919e+01,
   1.4698e+01,  1.4925e+01]],
 [[-3.5113e+01, -3.8697e+01, -3.3207e+01,  …, -3.3933e+01,
   -3.0985e+01, -1.3111e+01],
  [-3.5657e+01, -4.4277e+01, -3.8697e+01,  …, -3.9604e+01,
   -3.9150e+01, -2.1549e+01],
  [-4.0239e+01, -5.1081e+01, -4.5320e+01,  …, -4.6545e+01,
   -4.7180e+01, -2.4044e+01],
  …,
  [-1.1251e+01, -6.1243e+00, -5.0355e+00,  …, -4.9584e+01,
   -4.5728e+01, -2.1957e+01],
  [-2.4951e+00, -1.2249e+00,  1.8146e-01,  …, -7.1042e+01,
   -6.6097e+01, -3.3343e+01],
  [-6.6233e+00, -7.1224e+00, -5.9429e+00,  …, -5.7841e+01,
   -6.1379e+01, -3.9196e+01]],

 [[-2.6766e+00, -9.9350e+00, -6.9863e+00,  …, -6.0336e+00,
   -1.1523e+01, -1.1341e+01],
  [-4.5365e-01, -1.5243e+01, -1.0116e+01,  …, -1.2113e+01,
   -1.6831e+01, -4.8541e+00],
  [-7.9389e+00, -1.5969e+01, -9.1184e+00,  …, -8.0297e+00,
   -1.3973e+01, -8.0297e+00],
  …,
  [-5.2170e+00, -1.1069e+01, -1.3474e+01,  …, -9.2092e+00,
   -1.1976e+01, -6.2150e+00],
  [ 4.2190e+00,  6.3511e+00,  6.8048e+00,  …, -2.9714e+01,
   -4.0239e+01, -2.5178e+01],
  [-9.5267e-01, -8.8009e+00, -1.2566e+01,  …,  5.3168e+01,
    5.3622e+01,  3.4296e+01]]],


[[[-3.8969e+01, -6.7413e+01, -6.1742e+01,  …, -6.2922e+01,
   -6.3784e+01, -4.1328e+01],
  [ 4.1010e+01,  4.6862e+01,  4.8904e+01,  …,  5.7387e+01,
    5.1172e+01,  2.9533e+01],
  [ 1.3791e+01,  8.5740e+00,  8.6648e+00,  …,  5.4439e-01,
    4.2643e+00,  8.6648e+00],
  …,
  [ 7.7121e-01,  1.0888e+00, -3.9921e+00,  …,  8.5287e+00,
    5.2170e+00,  1.0071e+01],
  [-2.0414e+00, -4.1282e+00, -8.6194e+00,  …, -7.1677e+00,
   -1.0842e+01, -7.4853e+00],
  [ 7.7575e+00,  9.0731e+00,  8.8462e+00,  …,  4.8541e+00,
    1.3610e-01, -9.0731e-01]],

 [[ 4.0828e-01, -3.2981e+01, -1.3700e+01,  …, -8.2565e+00,
   -1.9053e+00, -3.1710e+01],
  [-3.3026e+01, -1.5379e+01, -5.9429e+00,  …,  3.7653e+00,
   -9.5721e+00, -3.1756e-01],
 [-1.4381e+01, -6.1697e+00,  4.1736e+00,  …, -9.0730e-01,
  -1.9598e+01, -3.3570e+00],
 …,
 [ 1.1704e+01, -5.6253e+00, -6.9863e+00,  …, -3.0395e+01,
   3.0667e+01,  1.0978e+01],
 [ 1.1341e+01, -1.5651e+01, -3.1756e-01,  …,  2.0959e+01,
   2.4906e+01, -5.0355e+00],
 [ 1.1069e+01, -2.0097e+01, -9.9804e-01,  …,  3.9286e+01,
   4.5365e-02,  9.0730e-02]],

[[ 5.5799e+00, -6.8048e-01, -2.3590e+00,  …, -6.5780e+00,
   4.3097e+00,  1.3020e+01],
 [-6.8048e+00, -5.0355e+00, -4.2643e+00,  …, -3.4931e+00,
   1.1795e+01,  1.2249e+00],
 [ 2.2683e-01, -2.0868e+00, -6.3511e+00,  …, -1.3610e+00,
   1.5424e+01, -5.2624e+00],
 …,
 [-1.3655e+01, -1.4154e+01, -1.6150e+01,  …, -2.7219e-01,
  -3.2527e+01, -1.5878e+01],
 [-2.0777e+01, -2.4724e+01, -1.9190e+01,  …, -4.1101e+01,
  -3.8288e+01, -1.7148e+01],
 [-1.4472e+01,  7.2584e-01,  8.1657e-01,  …, -1.5152e+01,
  -1.0389e+01,  1.1795e+01]],


…,


[[ 2.0868e+00,  1.4880e+01,  1.7284e+01,  …,  1.0752e+01,
  -2.7219e-01,  1.4562e+01],
 [ 2.0233e+01,  3.2663e+01,  3.2572e+01,  …,  2.1821e+01,
   1.2339e+01,  2.8535e+01],
 [ 2.0550e+01,  2.8036e+01,  2.2184e+01,  …,  1.5560e+01,
   1.3973e+01,  2.6493e+01],
 …,
 [ 1.7466e+01,  1.6876e+01,  2.5677e+01,  …,  4.2326e+01,
   3.7426e+01,  1.7692e+01],
 [ 1.7103e+01,  2.0596e+01,  2.6766e+01,  …,  3.1846e+01,
   2.2728e+01,  1.3292e+01],
 [ 1.5424e+01,  2.6267e+01,  2.9125e+01,  …,  2.9079e+01,
   2.9533e+01,  1.9235e+01]],

[[-3.4568e+01, -3.1892e+01, -2.6312e+01,  …, -3.5748e+01,
  -3.3888e+01, -1.3247e+01],
 [-3.9468e+01, -4.2643e+01, -3.4704e+01,  …, -4.5184e+01,
  -4.6636e+01, -2.5042e+01],
 [-3.8061e+01, -4.3959e+01, -3.1846e+01,  …, -5.4438e+01,
  -5.4438e+01, -2.7900e+01],
 …,
 [-2.6085e+01, -3.5430e+01, -4.1146e+01,  …, -4.1781e+01,
   -5.1807e+01, -4.6998e+01],
  [-1.8010e+01, -2.8898e+01, -3.6927e+01,  …, -3.1484e+01,
   -4.2144e+01, -4.2779e+01],
  [-2.8580e+00, -1.1477e+01, -1.5333e+01,  …, -1.3383e+01,
   -1.7375e+01, -2.3726e+01]],

 [[-3.0395e+00, -7.3492e+00, -5.8974e-01,  …, -9.5267e+00,
   -1.3746e+01, -1.5016e+01],
  [-4.0375e+00, -1.1069e+01, -1.8600e+00,  …, -1.4789e+01,
   -1.4290e+01, -3.6746e+00],
  [-2.7219e-01, -8.6194e+00, -3.8107e+00,  …, -3.0395e+00,
   -1.1750e+01, -7.4399e+00],
  …,
  [-1.0434e+00, -7.2131e+00, -1.1750e+01,  …, -1.1387e+01,
   -6.2150e+00, -8.0297e+00],
  [ 2.0959e+01,  2.5495e+01,  2.5677e+01,  …,  3.8107e+01,
    3.4704e+01,  1.5152e+01],
  [-1.3383e+01, -2.1322e+01, -2.2410e+01,  …, -3.9740e+01,
   -3.0576e+01, -2.4815e+01]]],


…,


[[[-1.2158e+01, -1.8055e+01, -1.4562e+01,  …,  1.7239e+00,
    2.5405e+00,  3.9468e+00],
  [ 6.3965e+00,  3.9014e+00,  7.1677e+00,  …, -1.6332e+01,
   -2.9941e+01, -1.6059e+01],
  [ 5.9429e+00, -4.9902e-01,  1.7239e+00,  …,  4.1736e+00,
    5.7614e+00,  8.7101e+00],
  …,
  [ 9.0731e-01, -2.5858e+00,  4.7180e+00,  …,  5.7160e+00,
    9.5267e-01,  6.5326e+00],
  [ 2.2683e-01, -4.0375e+00,  6.8502e+00,  …,  2.3771e+01,
    7.2584e+00, -4.0375e+00],
  [-7.8936e+00, -2.2819e+01, -1.5470e+01,  …, -1.2793e+01,
    7.9389e+00,  6.1243e+00]],

 [[ 3.9921e+00, -7.4399e+00, -7.7575e+00,  …,  5.8521e+00,
   -5.8975e-01,  1.0116e+01],
  [-1.4426e+01,  1.3610e+00,  3.4931e+00,  …,  3.1257e+01,
    1.4971e+00, -1.3111e+01],
  [-1.2249e+00,  1.3973e+01, -5.3077e+00,  …, -2.0687e+01,
   -1.0978e+01, -6.8955e+00],
  …,
  [ 8.6193e-01,  1.4925e+01, -1.1750e+01,  …, -1.4245e+01,
    8.0297e+00,  9.0730e-01],
  [ 8.9370e+00,  1.0434e+01, -2.0278e+01,  …,  1.3247e+01,
   8.6648e+00, -2.0142e+01],
  [ 2.0414e+00,  1.7692e+00, -7.2584e+00,  …, -6.1243e+00,
   -1.7103e+01,  5.5346e+00]],

 [[ 2.7990e+01,  2.7219e+01,  1.5606e+01,  …,  1.4245e+01,
   -4.8995e+00, -2.7945e+01],
  [ 2.1412e+01,  2.7673e+00, -1.2475e+01,  …, -8.0297e+00,
   -2.9714e+01, -2.9624e+01],
  [ 1.4834e+01, -1.4517e+00, -6.2150e+00,  …, -1.6377e+01,
   -4.0239e+01, -3.9785e+01],
  …,
  [ 1.4971e+01, -4.0829e-01,  2.8126e+00,  …, -4.0103e+01,
   -3.1529e+01, -6.0336e+00],
  [ 2.0505e+01,  9.3906e+00, -3.9014e+00,  …, -3.5385e+01,
   -2.0550e+01, -1.0434e+01],
  [ 1.1114e+01, -7.1677e+00, -2.7083e+01,  …,  1.1750e+01,
   -2.5405e+00,  1.2249e+00]],


 …,

 [[-6.8955e+00,  9.0732e-02, -5.4438e+00,  …, -2.6312e+00,
    1.2067e+01,  1.1931e+01],
  [ 2.8580e+00,  8.5287e+00,  4.2643e+00,  …, -5.0355e+00,
    1.3020e+01,  1.0071e+01],
  [ 3.9468e+00,  1.1205e+01,  1.0978e+01,  …,  8.9370e+00,
    2.1685e+01,  1.7239e+01],
  …,
  [ 1.3247e+01,  1.0933e+01,  1.1251e+01,  …,  1.1840e+01,
    1.2702e+01,  1.1341e+01],
  [ 8.3926e+00,  7.8482e+00,  1.4426e+01,  …,  2.4361e+01,
    9.8443e+00,  9.4813e+00],
  [ 7.4853e+00,  1.2385e+01,  1.9734e+01,  …,  2.2683e+01,
    1.9961e+01,  1.0842e+01]],

 [[-1.6422e+01, -2.3227e+01, -2.6267e+01,  …, -1.3337e+01,
   -3.1574e+01, -2.8762e+01],
  [-1.8328e+01, -2.5314e+01, -2.6947e+01,  …, -1.5742e+01,
   -2.9760e+01, -3.2572e+01],
  [-2.2909e+01, -2.8308e+01, -3.6201e+01,  …, -2.3272e+01,
   -2.8353e+01, -2.6130e+01],
  …,
  [-5.7705e+01, -6.2150e+01, -4.6998e+01,  …, -7.6985e+01,
   -4.9131e+01, -2.5677e+01],
  [-5.5709e+01, -6.4056e+01, -4.8949e+01,  …, -6.9681e+01,
   -6.2876e+01, -3.8016e+01],
  [-4.3460e+01, -5.4756e+01, -4.7996e+01,  …, -2.9170e+01,
   -2.5994e+01, -1.9961e+01]],
 [[-4.8541e+00, -3.6746e+00,  1.6785e+00,  …, -1.5243e+01,
   -2.3272e+01, -1.5606e+01],
  [-2.0414e+00, -1.4472e+01, -1.7057e+01,  …, -1.8872e+01,
   -6.0336e+00, -6.4872e+00],
  [-1.4608e+01, -1.6422e+01, -1.2339e+01,  …, -5.1716e+00,
   -4.2190e+00, -6.9409e+00],
  …,
  [-6.2604e+00, -1.0162e+01, -4.5819e+00,  …,  2.2184e+01,
   -5.3077e+00, -1.2974e+01],
  [-1.2566e+01, -2.1276e+01, -2.5858e+01,  …,  1.1704e+01,
    2.0006e+01,  1.3065e+01],
  [ 1.9734e+01,  2.6040e+01,  3.4750e+01,  …, -2.1639e+01,
   -1.4744e+01, -9.4360e+00]]],


[[[-3.5385e+01, -6.4328e+01, -6.0608e+01,  …, -6.0018e+01,
   -6.2695e+01, -4.0375e+01],
  [ 3.7835e+01,  4.2008e+01,  4.8223e+01,  …,  4.8722e+01,
    4.3868e+01,  2.6130e+01],
  [ 8.5740e+00, -5.4438e-01,  3.3570e+00,  …,  8.1658e-01,
    6.8047e-01,  2.6312e+00],
  …,
  [ 4.6726e+00, -7.7121e-01,  7.2585e-01,  …, -3.2663e+00,
   -3.0395e+00,  2.6312e+00],
  [ 9.4813e+00,  4.9902e+00,  2.4497e+00,  …,  7.3038e+00,
    3.1302e+00, -1.7239e+00],
  [ 1.2929e+01,  1.2929e+01,  1.3519e+01,  …,  1.2748e+01,
    1.3020e+01,  7.5306e+00]],

 [[ 1.3844e-06, -2.3635e+01, -1.1523e+01,  …, -1.3973e+01,
   -8.9823e+00, -2.4497e+01],
  [-3.0123e+01, -1.2385e+01, -8.9823e+00,  …, -6.4419e+00,
   -7.1677e+00,  2.7219e+00],
  [-1.2884e+01, -4.7180e+00, -4.4458e+00,  …, -4.8541e+00,
   -6.5780e+00, -2.2229e+00],
  …,
  [-1.2521e+01, -1.8146e-01, -4.0829e+00,  …, -1.5606e+01,
    4.1282e+00,  2.6312e+00],
  [-5.8975e+00, -2.1775e+00, -8.5740e+00,  …,  7.2584e+00,
    6.1697e+00, -1.0525e+01],
  [-1.3428e+01, -2.9941e+00, -2.6312e+00,  …,  1.5424e+00,
    8.6194e-01,  2.4951e+00]],

 [[ 5.6253e+00, -8.6648e+00, -6.4872e+00,  …, -4.9448e+00,
   -9.0277e+00,  2.3136e+00],
  [-7.9843e+00, -1.1160e+01, -6.4419e+00,  …, -5.7160e+00,
   -7.3492e+00,  1.1795e+00],
  [-3.9468e+00, -7.5760e+00, -2.2683e+00,  …, -1.7239e+00,
  -3.9014e+00,  9.0730e-01],
 …,
 [-1.0661e+01, -1.5016e+01, -2.6312e+00,  …, -9.6628e+00,
  -2.2864e+01, -8.1204e+00],
 [-1.2702e+01, -2.6312e+00,  3.3570e+00,  …, -1.7647e+01,
  -1.1341e+01, -1.4971e+00],
 [-3.8107e+00, -1.5878e+00, -2.4497e+00,  …,  6.5780e+00,
   2.9487e+00,  9.2999e+00]],

…,

[[ 9.9804e-01,  1.2566e+01,  9.3453e+00,  …,  8.7101e+00,
   6.8048e+00,  1.0116e+01],
 [ 1.9235e+01,  3.1302e+01,  2.7854e+01,  …,  2.4044e+01,
   2.1821e+01,  2.1730e+01],
 [ 1.8917e+01,  2.6992e+01,  2.2093e+01,  …,  1.7692e+01,
   1.6377e+01,  1.6332e+01],
 …,
 [ 1.3156e+01,  1.8282e+01,  1.3428e+01,  …,  1.6105e+01,
   1.8509e+01,  1.0888e+01],
 [ 1.6513e+01,  2.3545e+01,  2.1503e+01,  …,  2.0188e+01,
   1.7239e+01,  1.1114e+01],
 [ 1.2113e+01,  1.3247e+01,  1.3337e+01,  …,  1.5333e+01,
   1.3882e+01,  8.8462e+00]],

[[-3.6564e+01, -3.9422e+01, -3.1166e+01,  …, -3.2572e+01,
  -2.7355e+01, -9.5721e+00],
 [-3.9286e+01, -5.0492e+01, -4.3732e+01,  …, -4.1918e+01,
  -4.0375e+01, -2.0414e+01],
 [-4.0012e+01, -5.2896e+01, -4.4322e+01,  …, -4.1781e+01,
  -4.0965e+01, -2.0823e+01],
 …,
 [-3.2708e+01, -4.3415e+01, -3.6292e+01,  …, -4.2643e+01,
  -4.1419e+01, -2.8580e+01],
 [-4.1010e+01, -5.6162e+01, -5.1036e+01,  …, -4.1963e+01,
  -4.4594e+01, -3.4296e+01],
 [-2.5450e+01, -3.8651e+01, -3.6701e+01,  …, -2.0097e+01,
  -1.9235e+01, -1.7965e+01]],

[[ 8.6194e-01, -3.6746e+00, -2.5858e+00,  …, -1.2702e+00,
  -4.7634e+00, -5.6253e+00],
 [-2.1775e+00, -1.1659e+01, -6.7141e+00,  …, -5.5799e+00,
  -1.0661e+01, -6.1697e+00],
 [-3.5385e+00, -1.2612e+01, -8.2565e+00,  …, -9.8896e+00,
  -1.4562e+01, -9.4360e+00],
 …,
 [-5.1263e+00, -1.6195e+01, -1.5515e+01,  …, -3.9921e+00,
  -1.1114e+01, -1.2475e+01],
  [-5.8521e+00, -1.3610e+01, -8.8916e+00,  …,  1.7511e+01,
    1.8917e+01,  1.0207e+01],
  [ 2.7355e+01,  2.6947e+01,  2.5858e+01,  …, -8.6194e+00,
   -7.8482e+00, -7.8028e+00]]],


[[[-1.7239e+00,  4.7634e+00,  3.4931e+00,  …,  2.5858e+00,
   -2.7673e+00,  5.8975e+00],
  [-2.3590e+01, -3.7971e+01, -4.1419e+01,  …, -3.7109e+01,
   -3.8424e+01, -1.4426e+01],
  [ 1.1477e+01,  1.0026e+01,  9.5267e+00,  …, -3.3117e+00,
   -5.8068e+00,  6.8048e-01],
  …,
  [ 2.1186e+01,  2.2819e+01,  2.5268e+01,  …, -1.5424e+00,
   -5.9882e+00,  1.8600e+00],
  [ 1.5878e+00, -5.7160e+00, -1.2385e+01,  …, -8.7101e+00,
   -9.9350e+00, -3.8561e+00],
  [-1.0434e+01, -1.5243e+01,  1.3156e+00,  …,  7.0770e+00,
    2.6312e+00,  6.3511e-01]],

 [[ 1.7874e+01,  3.0395e+00,  9.4813e+00,  …,  3.3570e+00,
   -8.1658e-01,  5.9429e+00],
  [ 2.1367e+01,  1.8146e-01,  1.7692e+01,  …,  7.4853e+00,
    4.1282e+00, -1.5560e+01],
  [-3.9921e+00,  5.2170e+00, -8.6194e+00,  …, -4.5365e+00,
   -8.3018e+00, -1.5878e+00],
  …,
  [-2.4316e+01,  1.7511e+01,  1.4744e+01,  …, -5.4438e-01,
   -1.4335e+01, -1.3610e+00],
  [ 1.9779e+01,  2.6720e+01, -1.2612e+01,  …, -1.4517e+00,
   -1.3383e+01,  6.5780e+00],
  [-6.3511e+00, -1.8373e+01,  1.2702e+00,  …,  4.9902e-01,
    2.1322e+00,  8.3926e+00]],

 [[ 1.3844e-06, -2.6312e+00, -1.7103e+01,  …, -1.8191e+01,
   -1.8600e+01, -2.8217e+01],
  [ 3.0395e+00, -1.0752e+01, -5.2624e+00,  …, -2.3908e+01,
   -1.2475e+01, -8.8462e+00],
  [-7.2584e+00, -1.2385e+01, -1.0162e+01,  …, -8.4379e+00,
   -6.6687e+00, -8.9823e+00],
  …,
  [-4.5363e-02, -2.5087e+01, -2.0687e+01,  …, -1.6105e+01,
   -1.1795e+01, -1.1114e+01],
  [ 7.3945e+00,  1.8509e+01,  1.1523e+01,  …, -3.2209e+01,
   -3.0622e+01, -1.8600e+01],
  [ 1.2566e+01, -1.1750e+01, -2.3454e+01,  …, -1.0434e+01,
   -1.0071e+01,  1.2067e+01]],
  …,

  [[ 8.7555e+00,  1.4789e+01,  1.3836e+01,  …,  2.9125e+01,
     3.4342e+01,  2.0188e+01],
   [ 5.5799e+00,  9.9350e+00,  2.2229e+00,  …,  2.3817e+01,
     3.2799e+01,  2.2320e+01],
   [ 1.1795e+01,  1.0615e+01,  1.1296e+01,  …,  3.0032e+01,
     3.7336e+01,  2.4225e+01],
   …,
   [ 1.9144e+01,  1.9689e+01,  1.8418e+01,  …,  2.2819e+01,
     3.6020e+01,  2.3998e+01],
   [ 8.8916e+00,  7.9843e+00,  6.1243e+00,  …,  2.0777e+01,
     3.3162e+01,  1.8509e+01],
   [ 5.1716e+00,  1.2793e+01,  1.7511e+01,  …,  3.1665e+01,
     3.5793e+01,  2.0278e+01]],

  [[-1.8146e+01, -2.5767e+01, -3.5067e+01,  …, -5.3667e+01,
    -6.3920e+01, -4.6590e+01],
   [-2.4724e+01, -3.1892e+01, -4.1192e+01,  …, -3.8833e+01,
    -5.4756e+01, -4.7180e+01],
   [-1.8917e+01, -2.6403e+01, -3.4296e+01,  …, -2.7310e+01,
    -4.7135e+01, -4.4186e+01],
   …,
   [-4.3052e+01, -5.7478e+01, -5.2261e+01,  …, -3.7426e+01,
    -4.9176e+01, -4.5138e+01],
   [-4.0647e+01, -5.4438e+01, -5.8430e+01,  …, -2.8399e+01,
    -4.0919e+01, -4.0557e+01],
   [-3.5022e+01, -4.0647e+01, -3.6701e+01,  …, -1.0615e+01,
    -1.7375e+01, -2.3545e+01]],

  [[-1.7829e+01, -2.9170e+01, -3.4841e+01,  …, -1.4426e+01,
    -1.0026e+01, -2.9487e+00],
   [-4.7634e+00, -6.5326e+00, -7.3038e+00,  …, -1.8509e+01,
    -1.3247e+01, -1.5197e+01],
   [ 1.0253e+01,  1.3791e+01,  4.4458e+00,  …, -1.4154e+01,
    -2.2683e+00, -4.2643e+00],
   …,
   [-5.3077e+00, -3.0939e+01, -5.1898e+01,  …, -1.0525e+01,
    -1.3610e+00, -6.2604e+00],
   [-2.7764e+01, -1.6241e+01,  1.5742e+01,  …,  3.6927e+01,
     3.4523e+01,  1.2339e+01],
   [ 3.2073e+01,  2.1412e+01,  4.1282e+00,  …, -3.7472e+01,
    -3.0803e+01, -2.3726e+01]]]], device='cuda:0',
grad_fn=<MulBackward0>)
```

[7]:

```
## 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 - output_recovered).mean())
```

```
tensor(-0.2005, device='cuda:0', grad_fn=<MeanBackward0>)
```

[24]:

```
# 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 = 16 # 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(4) # range(0, 4)   need to use icg
oc_tileg = range(4) # range(0, 4)   need to use ocg

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(njg)+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))  ## mergin ni and nj index into nij
# a_pad.size() = [64, (32+2pad)*(32+2pad)]


###### a_tile is "tiled version of a_pad"
# generate zero vector of size[ic tile num, array_size, (32+2pad)*(32+2pad)]
```

```
a_tile = torch.zeros(4, array_size, a_pad.size(1)).cuda()

# then, embed a_pad into a_tile at right position
for ic_tile in ic_tileg:
    a_tile[ic_tile] = a_pad[array_size*ic_tile:array_size*(ic_tile+1), :]

###### w_tile is "tiled version of w_int"
## generate zero vector of size [ic tile num, oc tile num, array_size,␣
↪array_size, ki_dim*ki_dim]
w_tile = torch.zeros(4, 4, array_size, array_size, ki_dim*ki_dim).cuda() ##␣
↪tiled version of w

# then, embed w_int into w_tile at right position
for oc_tile in oc_tileg:
    for ic_tile in ic_tileg:
        w_tile[ic_tile, oc_tile, :, :, :] = w_int[array_size*oc_tile:
↪array_size*(oc_tile+1), array_size*ic_tile:array_size*(ic_tile+1), :]
```

[28]:

```
p_nijg = range(a_pad.size(1)) ## paded activation's nij group

## generate zero vector of size [ic tile num, oc tile num, array_size,␣
↪len(p_nijg), ki_dim*ki_dim]
psum = torch.zeros(4, 4, 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_tile[ic_tile, oc_tile, :, :,␣
↪kij])
                psum[ic_tile, oc_tile, :, nij, kij] = m(a_tile[ic_tile, :,␣
↪nij]).cuda()
```

[29]:

```
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()

print(out.size())
```

```
### 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[array_size*oc_tile:array_size*(oc_tile+1),o_nij] =␣
 ↪out[array_size*oc_tile:array_size*(oc_tile+1),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]
```

```
torch.Size([64, 1024])
```

[30]:

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

```
tensor(-0.0064, device='cuda:0', grad_fn=<SumBackward0>)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: