project_part1

December 14, 2023

[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 = 64
model_name = "VGG16_quant"
model = VGG16_quant()

device = torch.device("cuda" if use_gpu else "cpu")
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 = [35, 50]
    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)
print (model)
```

```
=> Building model…
0
3
6
7
10
13
14
17
20
23
24
27
29
32
33
36
39
42
Files already downloaded and verified
Files already downloaded and verified
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, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (25): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): QuantConv2d(
      8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (28): ReLU(inplace=True)
    (29): QuantConv2d(
      8, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (30): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (31): ReLU(inplace=True)
    (32): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (33): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (34): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (35): ReLU(inplace=True)
    (36): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (37): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (38): ReLU(inplace=True)
    (39): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (40): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (41): ReLU(inplace=True)
    (42): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (43): AvgPool2d(kernel_size=1, stride=1, padding=0)
  )
  (classifier): Linear(in_features=512, out_features=10, bias=True)
)
```

[2]:
```
# # This cell is from the website

# lr = 4e-2
# weight_decay = 1e-4
# epochs = 60
# best_prec = 0

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


# if not os.path.exists('result_temp'):
#     os.makedirs('result_temp')

# fdir = 'result_temp/'+str(model_name)

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


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

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

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

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

```
#         'best_prec': best_prec,
#         'optimizer': optimizer.state_dict(),
#     }, is_best, fdir)
```

[3]:
```
fdir = 'result/'+str(model_name)+'/model_best.pth.tar'

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


criterion = nn.CrossEntropyLoss().cuda()

model.eval()
model.cuda()


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

```
Test: [0/157]   Time 1.442 (1.442)      Loss 0.3054 (0.3054)    Prec 89.062%
(89.062%)
Test: [100/157] Time 0.013 (0.027)      Loss 0.2183 (0.3487)    Prec 92.188%
(90.996%)
 * Prec 91.420%
```

[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 name, layer in model.named_modules():

    if isinstance(layer, torch.nn.Conv2d):
        print(i,"-th layer prehooked")
        layer.register_forward_pre_hook(save_output)
    i = i+1

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

dataiter = iter(testloader)
```

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

print("7st convolution's input size:", save_output.outputs[8][0].size())
print("7st convolution's input size:", save_output.outputs[9][0].size())
```

```
2 -th layer prehooked
6 -th layer prehooked
11 -th layer prehooked
15 -th layer prehooked
20 -th layer prehooked
24 -th layer prehooked
28 -th layer prehooked
33 -th layer prehooked
37 -th layer prehooked
40 -th layer prehooked
45 -th layer prehooked
49 -th layer prehooked
53 -th layer prehooked
7st convolution's input size: torch.Size([64, 8, 4, 4])
7st convolution's input size: torch.Size([64, 8, 4, 4])
```

[5]:
```
weight_q = model.features[27].weight_q
w_alpha = model.features[27].weight_quant.wgt_alpha
w_bit = 4

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

```
tensor([[[[-4.0000, -7.0000, -7.0000],
          [-4.0000, -5.0000, -7.0000],
          [-3.0000, -5.0000, -5.0000]],

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

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

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

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

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

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

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

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


[[[-5.0000,  1.0000, -1.0000],
  [-7.0000, -7.0000, -7.0000],
  [-7.0000, -7.0000, -7.0000]],

 [[-2.0000, -7.0000, -7.0000],
  [ 7.0000,  7.0000,  4.0000],
  [ 7.0000,  7.0000, -7.0000]],

 [[ 4.0000,  4.0000,  0.0000],
  [-7.0000,  7.0000, -3.0000],
  [-7.0000,  0.0000, -6.0000]],

 [[ 7.0000,  7.0000,  7.0000],
  [ 7.0000,  7.0000,  7.0000],
  [ 7.0000,  7.0000,  7.0000]],

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

 [[ 0.0000,  5.0000,  4.0000],
  [-2.0000,  3.0000,  6.0000],
  [ 4.0000,  0.0000,  0.0000]],

 [[-0.0000,  0.0000, -2.0000],
  [-6.0000,  7.0000,  0.0000],
  [ 7.0000,  7.0000,  1.0000]],

 [[ 7.0000, -3.0000, -2.0000],
  [ 7.0000, -0.0000, -1.0000],
  [ 7.0000,  4.0000,  6.0000]]],
```

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

 [[ 7.0000,  7.0000, -7.0000],
  [ 7.0000,  7.0000,  7.0000],
  [-7.0000,  3.0000, -7.0000]],

 [[-7.0000, -7.0000, -7.0000],
  [-7.0000,  0.0000, -6.0000],
  [-4.0000, -1.0000, -3.0000]],

 [[-3.0000, -4.0000, -3.0000],
  [ 3.0000, -7.0000, -6.0000],
  [ 3.0000,  1.0000,  0.0000]],

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

 [[-0.0000,  1.0000,  4.0000],
  [-7.0000, -3.0000,  4.0000],
  [ 1.0000,  3.0000,  3.0000]],

 [[ 3.0000, -3.0000, -2.0000],
  [ 7.0000,  1.0000,  0.0000],
  [ 0.0000, -3.0000, -4.0000]],

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


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

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

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

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

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

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

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

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


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

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

 [[-7.0000,  3.0000, -3.0000],
  [ 2.0000,  7.0000, -7.0000],
  [ 7.0000,  7.0000,  7.0000]],

 [[ 1.0000, -1.0000,  7.0000],
  [ 7.0000,  2.0000,  1.0000],
  [ 2.0000, -7.0000,  7.0000]],

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

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

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

 [[-7.0000, -7.0000, -7.0000],
  [-7.0000,  7.0000, -7.0000],
```

```
  [-7.0000, -7.0000, -7.0000]]],


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

 [[-3.0000,  7.0000, -0.0000],
  [ 2.0000,  7.0000, -7.0000],
  [-7.0000, -7.0000,  2.0000]],

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

 [[ 1.0000,  7.0000,  7.0000],
  [ 7.0000,  7.0000,  7.0000],
  [-2.0000,  1.0000,  7.0000]],

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

 [[ 7.0000, -3.0000,  2.0000],
  [ 7.0000,  2.0000,  2.0000],
  [-7.0000, -3.0000, -3.0000]],

 [[-1.0000, -1.0000, -0.0000],
  [-2.0000, -6.0000, -2.0000],
  [ 1.0000, -5.0000, -4.0000]],

 [[-7.0000, -3.0000, -1.0000],
  [-4.0000, -3.0000, -6.0000],
  [-2.0000,  0.0000,  2.0000]]],


[[[ 7.0000, -0.0000,  7.0000],
  [ 7.0000,  7.0000,  7.0000],
  [ 7.0000,  7.0000,  7.0000]],

 [[ 7.0000, -1.0000, -7.0000],
  [ 7.0000, -7.0000, -7.0000],
  [-7.0000,  1.0000,  1.0000]],

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

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

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

 [[-5.0000,  3.0000, -5.0000],
  [ 7.0000,  7.0000,  7.0000],
  [-7.0000, -7.0000, -7.0000]],

 [[ 7.0000,  4.0000,  1.0000],
  [ 7.0000, -7.0000, -7.0000],
  [-7.0000, -7.0000, -7.0000]],

 [[ 7.0000,  1.0000, -7.0000],
  [ 7.0000,  1.0000, -7.0000],
  [ 7.0000,  7.0000,  7.0000]]],


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

 [[ 7.0000,  7.0000, -2.0000],
  [ 7.0000,  7.0000, -7.0000],
  [-7.0000, -7.0000, -1.0000]],

 [[ 1.0000,  7.0000, -7.0000],
  [-2.0000, -7.0000, -7.0000],
  [ 7.0000,  7.0000,  7.0000]],

 [[ 7.0000, -7.0000, -7.0000],
  [ 7.0000,  7.0000,  7.0000],
  [-4.0000, -0.0000,  7.0000]],

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

 [[ 2.0000,  1.0000,  1.0000],
  [ 7.0000,  7.0000,  7.0000],
  [ 7.0000,  7.0000,  7.0000]],

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

```
          [[ 7.0000,  0.0000,  0.0000],
           [-0.0000, -7.0000, -3.0000],
           [ 1.0000, -4.0000,  7.0000]]]], device='cuda:0',
       grad_fn=<DivBackward0>)
```

[6]:
```
act = save_output.outputs[8][0]
act_alpha  = model.features[27].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([[[[ 3.0000,  0.0000,  0.0000,  0.0000],
          [ 7.0000,  3.0000,  0.0000,  0.0000],
          [ 8.0000,  7.0000,  0.0000,  1.0000],
          [10.0000, 12.0000,  8.0000,  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],
          [ 1.0000,  1.0000,  3.0000,  4.0000]],

         [[ 1.0000,  1.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  1.0000,  2.0000],
          [ 4.0000,  7.0000,  7.0000,  6.0000]],

         …,

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

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

```
[[[ 3.0000,  0.0000,  0.0000,  0.0000],
  [ 4.0000,  0.0000,  0.0000,  0.0000],
  [ 5.0000,  5.0000,  0.0000,  1.0000],
  [12.0000, 15.0000, 11.0000,  6.0000]],

 [[ 0.0000,  0.0000,  2.0000,  2.0000],
  [ 0.0000,  0.0000,  3.0000,  8.0000],
  [ 0.0000,  5.0000, 15.0000, 15.0000],
  [10.0000, 15.0000, 15.0000, 15.0000]],

 [[ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 2.0000,  0.0000,  0.0000,  0.0000],
  [ 7.0000,  0.0000,  0.0000,  0.0000],
  [ 9.0000,  2.0000,  0.0000,  1.0000]],

 …,

 [[ 1.0000,  0.0000,  5.0000,  5.0000],
  [ 3.0000,  0.0000,  7.0000,  8.0000],
  [ 0.0000,  0.0000,  2.0000,  5.0000],
  [ 2.0000,  0.0000,  2.0000,  3.0000]],

 [[ 1.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 3.0000,  0.0000,  0.0000,  0.0000],
  [ 6.0000,  0.0000,  0.0000,  0.0000]],

 [[ 8.0000, 14.0000,  4.0000,  0.0000],
  [10.0000, 15.0000,  4.0000,  2.0000],
  [ 8.0000,  7.0000,  2.0000,  4.0000],
  [ 4.0000,  2.0000,  2.0000,  2.0000]]],


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

 [[ 0.0000,  0.0000,  5.0000,  6.0000],
  [ 0.0000,  0.0000,  6.0000,  9.0000],
  [ 1.0000,  4.0000, 13.0000, 12.0000],
  [ 7.0000, 12.0000, 15.0000, 12.0000]],

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

```
 …,

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

 [[ 3.0000,  7.0000,  3.0000,  0.0000],
  [ 6.0000, 13.0000,  7.0000,  1.0000],
  [ 4.0000, 10.0000,  7.0000,  5.0000],
  [ 1.0000,  4.0000,  4.0000,  3.0000]]],


…,


[[[ 4.0000,  2.0000,  0.0000,  2.0000],
  [ 6.0000,  3.0000,  2.0000,  3.0000],
  [ 5.0000,  3.0000,  2.0000,  2.0000],
  [ 7.0000,  8.0000,  7.0000,  5.0000]],

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

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

 …,

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

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

```
 [[ 2.0000,  2.0000,  0.0000,  0.0000],
  [ 6.0000,  7.0000,  1.0000,  0.0000],
  [ 7.0000,  9.0000,  6.0000,  0.0000],
  [ 1.0000,  1.0000,  0.0000,  0.0000]]],


[[[ 1.0000,  0.0000,  0.0000,  0.0000],
  [ 3.0000,  1.0000,  0.0000,  0.0000],
  [ 1.0000,  0.0000,  0.0000,  2.0000],
  [ 7.0000,  9.0000,  9.0000,  7.0000]],

 [[ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  2.0000],
  [ 2.0000,  2.0000,  7.0000,  8.0000]],

 [[ 2.0000,  3.0000,  0.0000,  0.0000],
  [ 4.0000,  2.0000,  0.0000,  0.0000],
  [ 5.0000,  2.0000,  0.0000,  0.0000],
  [ 7.0000,  8.0000,  5.0000,  3.0000]],

 …,

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

 [[ 2.0000,  0.0000,  0.0000,  0.0000],
  [ 4.0000,  0.0000,  0.0000,  0.0000],
  [ 8.0000,  9.0000,  0.0000,  0.0000],
  [ 7.0000,  7.0000,  1.0000,  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]]],


[[[ 2.0000,  0.0000,  0.0000,  0.0000],
  [ 3.0000,  4.0000,  0.0000,  2.0000],
  [ 8.0000,  9.0000,  0.0000,  1.0000],
  [10.0000, 15.0000, 10.0000,  7.0000]],

 [[ 0.0000,  0.0000,  3.0000,  2.0000],
  [ 0.0000,  0.0000,  7.0000,  3.0000],
  [ 0.0000,  0.0000,  1.0000,  0.0000],
```

```
          [ 5.0000,  8.0000,  7.0000,  7.0000]],

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

         …,

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

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

[7]:
```
conv_int = torch.nn.Conv2d(in_channels = 8, out_channels=8, kernel_size = 3,␣
 ↪padding=1, bias=False)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
psum_int = conv_int(act_int)

psum_recovered = psum_int * (act_alpha / (2**act_bit-1)) * (w_alpha /␣
 ↪(2**(w_bit-1)-1))
relu = model.features[28]
psum_recovered = relu(psum_recovered)
print(psum_recovered)
```

```
tensor([[[[ 0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  0.0000]],

         [[ 0.0000,  0.0000,  0.0000,  0.0352],
          [ 0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  0.0000]],

         [[ 0.0000,  1.7965,  0.4579,  0.0000],
          [ 1.3034,  3.2408,  1.7965,  0.0000],
```

```
  [ 3.2056,  2.5010,  0.4227,  0.0000],
  [ 1.0920,  0.3170,  0.1057,  1.5147]],

 …,

 [[ 0.6341,  2.8533,  1.0568,  0.3170],
  [ 0.0000,  3.9453,  2.3954,  0.3523],
  [ 0.0000,  4.3680,  4.1919,  0.0000],
  [ 0.0000,  0.3523,  1.0216,  1.1625]],

 [[ 4.9316,  5.6009,  2.1488,  0.4932],
  [ 6.4816,  8.3838,  2.9238,  0.0000],
  [ 8.1020, 11.9064,  7.0452,  0.0705],
  [ 9.0179, 12.5757,  9.7576,  3.6987]],

 [[ 0.3170,  0.0000,  0.0000,  0.4932],
  [ 0.7750,  0.0000,  0.0000,  0.5988],
  [ 7.7145,  4.1567,  1.4090,  0.0000],
  [ 4.1919,  2.1136,  0.0000,  3.0999]]],


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

 [[ 2.7124, 10.7087, 20.1141, 17.9653],
  [ 0.0000,  8.7713, 27.9695, 26.1025],
  [ 3.3112, 10.6383, 20.8186, 20.4311],
  [ 9.8281,  6.4111,  3.8044,  3.0647]],

 [[ 0.0000,  0.2466,  0.0000,  0.6341],
  [ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  2.0783,  3.2408,  8.0315],
  [ 0.0000,  5.0021, 15.3586, 18.8107]],

 …,

 [[ 0.0000,  1.3386,  4.3680,  1.2329],
  [ 0.0000,  2.5010,  0.4579,  3.6283],
  [ 0.0000,  0.5284,  2.9238,  5.9180],
  [ 0.0000,  6.4816,  9.2292,  8.6304]],

 [[ 3.4874, 12.8223,  9.7576,  0.4932],
  [ 0.5284, 17.0142, 10.1451,  0.0000],
  [ 3.4522, 14.6540,  5.2487,  3.5226],
  [ 0.0000,  1.2329,  1.2681,  9.4054]],
```

```
 [[ 0.2114,  2.9590, 12.9984,  8.3133],
  [ 0.2818,  0.0000,  0.8102,  6.0941],
  [ 2.5363,  0.0000,  0.0000,  2.8181],
  [ 5.4248, 12.3643, 15.4642, 20.0436]]],


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

 [[ 2.9942,  7.7145, 16.2392, 17.5426],
  [ 4.3680,  8.2077, 22.0515, 23.2492],
  [ 2.9942,  9.9690, 19.3743, 21.3118],
  [ 6.0237,  4.8964,  5.7771,  6.5873]],

 [[ 0.0000,  0.0000,  2.2545,  3.6635],
  [ 0.0000,  0.0000,  0.0000,  1.1272],
  [ 0.0000,  0.0000,  0.8102,  6.6929],
  [ 0.0000,  3.6635,  9.5815, 13.7734]],

 …,

 [[ 0.0000,  1.6556,  2.3954,  0.0000],
  [ 0.0705,  4.0510,  2.4658,  2.1488],
  [ 0.0000,  0.0000,  0.0000,  1.1625],
  [ 0.0000,  2.9942,  4.4033,  5.1430]],

 [[ 3.6987,  9.1940,  5.8475,  0.0000],
  [ 2.1840, 11.9769,  8.1372,  2.1488],
  [ 0.0000,  1.0216,  1.1272,  4.7203],
  [ 0.0000,  0.0000,  0.0000,  9.4406]],

 [[ 6.8691,  5.2839,  7.1861,  5.6362],
  [ 2.9238,  0.0000,  1.2681,  7.4679],
  [ 1.9022,  0.0000,  0.0000,  5.3896],
  [ 1.9022,  6.4816, 11.8007, 18.0357]]],


…,


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

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

```
  [ 0.0000,  0.0000,  0.8807,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  0.0000]],

 [[ 0.0000,  2.1840,  2.8533,  1.3034],
  [ 0.0000,  0.0000,  1.1977,  2.6420],
  [ 0.7045,  0.4579,  0.9159,  0.0000],
  [ 0.4227,  0.7750,  1.3738,  0.5636]],

 …,

 [[ 0.0000,  1.3738,  0.1409,  0.3875],
  [ 0.0000,  0.1409,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  0.7397],
  [ 0.0000,  0.0000,  0.0000,  0.0000]],

 [[ 4.8612,  8.9826,  5.1430,  2.0079],
  [ 6.3759, 14.5836,  9.6167,  4.6851],
  [ 1.5499, 10.6735, 10.5678,  4.8964],
  [ 0.8807,  7.6088,  7.9611,  4.4737]],

 [[ 0.9511,  0.0000,  0.0000,  0.1057],
  [ 0.0000,  0.0000,  0.0000,  0.7750],
  [ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 1.9374,  1.3386,  2.8181,  2.8885]]],


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

 [[ 1.5499,  0.3170,  0.0000,  0.2466],
  [ 5.1078,  3.0647,  1.6204,  0.5636],
  [ 2.1136,  0.0000,  0.0000,  0.0000],
  [ 1.8670,  0.0000,  0.0000,  0.0000]],

 [[ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  0.0000],
  [ 0.0000,  0.0000,  0.2114,  0.7397],
  [ 0.0000,  2.1840,  6.9395,  6.7986]],

 …,

 [[ 0.0000,  0.3523,  0.9511,  0.5988],
  [ 0.0000,  0.0000,  1.1272,  0.5988],
  [ 0.0000,  0.0000,  0.2114,  0.0000],
  [ 0.0000,  0.0000,  0.0000,  3.4522]],
```

```
          [[ 0.0000,  1.4795,  2.3249,  0.9863],
           [ 0.0000,  0.0000,  0.0000,  0.8454],
           [ 0.0000,  0.5284,  4.4385,  1.1272],
           [ 1.0568,  8.0668,  8.3133,  4.5089]],

          [[ 2.2545,  4.0862,  2.9942,  1.9727],
           [ 0.0000,  3.9101,  4.2271,  0.0000],
           [ 0.1057,  4.5442,  0.0705,  0.0000],
           [ 0.0000,  0.0000,  0.0000,  5.2135]]],


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

          [[ 0.0000,  0.0000,  1.3034,  3.3817],
           [ 0.0000,  0.0000,  0.0000,  1.5852],
           [ 0.0000,  0.0000,  0.0000,  0.0000],
           [ 4.3328,  1.4795,  0.0000,  0.0000]],

          [[ 0.6341,  0.0000,  2.3601,  0.0352],
           [ 0.7045,  2.6772,  5.3896,  3.6635],
           [ 0.0000,  2.7476,  2.9942,  2.6772],
           [ 0.0000,  3.2408,  7.5384,  6.0237]],

          …,

          [[ 0.4227,  0.8807,  0.0000,  0.0000],
           [ 0.9159,  2.1136,  4.4737,  0.7045],
           [ 0.8807,  4.8260,  6.3055,  0.0000],
           [ 0.0000,  5.2839,  4.5089,  4.4033]],

          [[ 2.3249,  1.5147,  0.3523,  0.0000],
           [ 6.2702,  6.6929,  1.6909,  2.5010],
           [10.8496, 14.5131, 10.7087,  3.6635],
           [ 2.6067,  8.2077,  9.9337,  4.8964]],

          [[ 2.8181,  0.0000,  0.0000,  0.0000],
           [ 3.4522,  0.0000,  0.3170,  4.6146],
           [ 8.0315,  0.4227,  0.0000,  0.0000],
           [ 6.4464, 10.5678,  4.6851,  5.9884]]]], device='cuda:0',
       grad_fn=<ReluBackward0>)
```

[8]:
```
difference = abs( save_output.outputs[9][0] - psum_recovered )
print(difference.mean())
```

```
tensor(2.8440e-07, device='cuda:0', grad_fn=<MeanBackward0>)
```

[23]:
```
print(len(icg))
print(oc_tileg)
print(w_int.size())
print(ic_tileg)
for i in ic_tileg:
    print(i)
```

```
8
range(0, 1)
torch.Size([8, 8, 9])
range(0, 1)
0
```

[94]:
```
# 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()
```

[95]:
```
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)
```

[96]:
```
# out_2D = torch.reshape(out, (out.size(0), o_ni_dim, -1)) # nij -> ni & nj
# difference = (out_2D - output_int[0,:,:,:])
# print(difference.abs().sum())
```

[97]:
```
### show this cell partially. The following cells should be printed by students␣
 ↪###
tile_id = 0
nij = 0 # just a random number
X = a_tile[tile_id,:,nij:nij+36]  # [tile_num, array row num, time_steps]

bit_precision = 4
file = open('activation_project.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(round(X[7-j,i].item()))
        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
```

[115]:
```
### show this cell partially. The following cells should be printed by students␣
 ↪###
tile_id = 0
nij = 0 # just a random number
X = a_tile[tile_id,:,nij:nij+36]  # [tile_num, array row num, time_steps]

bit_precision = 4
file = open('activation_project_int.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 #
        file.write(f'{int(X[7-j,i].item())},')
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close() #close file
```

[121]:
```
### 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

for kij_num in range(9):
    W = w_tile[tile_id,:,:,kij_num]  # w_tile[tile_num, array col num, array␣
 ↪row num, kij]

    file = open('weight_kij{}.txt'.format(kij_num), '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(1)):  # time step
        lis = []
        for j in range(W.size(0)): # row #
            if(int(W[i,j].item())>=0):
                W_bin = '{0:04b}'.format(round(W[i,j].item()))
            else:
                W_bin = '{0:04b}'.format(round(W[i,j].item())+16)
            lis.append(W_bin)
            #for k in range(bit_precision):
            #   file.write(W_bin[k])
            #file.write(' ')  # for visibility with blank between words, you␣
 ↪can use
        string_line = ''.join(reversed(lis))
        file.write(string_line)
        file.write('\n')
    file.close() #close file
```

[130]:
```
### 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

for kij_num in range(9):
    W = w_tile[tile_id,:,:,kij_num]  # w_tile[tile_num, array col num, array␣
 ↪row num, kij]

    file = open('weight_kij{}_int.txt'.format(kij_num), '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(1)):  # time step
        lis = []
        for j in range(W.size(0)): # row #
            lis.append(f'{round(W[i,j].item())},')
        string_line = ''.join(reversed(lis))
        file.write(string_line)
            #file.write(f'{int(W[i,j].item())},')
        file.write('\n')
    file.close() #close file
```

[123]:
```
### Complete this cell ###
ic_tile_id = 0
oc_tile_id = 0


kij = 0
nij = 0

bit_precision = 16

for kij_num in range(9):
    nij = 0
    psum_tile = psum[ic_tile_id,oc_tile_id,:,nij:nij+36,kij_num]
    psum_tile = torch.transpose(psum_tile,0,1)

    file = open('psum_kij{}.txt'.format(kij_num), '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(0)):  # time step
        lis = []
        for j in range(psum_tile.size(1)): # row #
            if(psum_tile[i,j].item() >= 0):
                psum_tile_bin = '{0:016b}'.format(round(psum_tile[i,j].item()))
            else:
                psum_tile_bin = '{0:016b}'.format(round(psum_tile[i,j].
 ↪item())+65536)
            lis.append(psum_tile_bin)
            #for k in range(bit_precision):
            #   file.write(psum_tile_bin[k])
            #file.write(' ')  # for visibility with blank between words, you␣
 ↪can use
        string_line = ''.join(reversed(lis))
        file.write(string_line)
        file.write('\n')
    file.close() #close file
```

[124]:
```
### Complete this cell ###
ic_tile_id = 0
oc_tile_id = 0


kij = 0
nij = 0

bit_precision = 16

for kij_num in range(9):
    nij = 0
    psum_tile = psum[ic_tile_id,oc_tile_id,:,nij:nij+36,kij_num]
    psum_tile = torch.transpose(psum_tile,0,1)

    file = open('psum_kij{}_int.txt'.format(kij_num), '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(0)):  # time step
        lis = []
        for j in range(psum_tile.size(1)): # row #
             lis.append(f'{round(psum_tile[i,j].item())},')
        string_line = ''.join(reversed(lis))
        file.write(string_line)
```

```
        #file.write(' ')  # for visibility with blank between words, you can␣
 ↪use
        file.write('\n')
file.close() #close file
```

[125]:
```
psum_tile
```

[125]:
```
tensor([[   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [ -26.0000,  -26.0000,    5.0000,  -11.0000,   18.0000,    1.0000,
            3.0000,   35.0000],
        [  -5.0000,   -6.0000,    0.0000,   -2.0000,    9.0000,   -1.0000,
           -4.0000,   14.0000],
        [  -3.0000,    0.0000,    3.0000,   -1.0000,    2.0000,   -3.0000,
           -7.0000,    7.0000],
        [  -3.0000,    0.0000,    3.0000,   -1.0000,    2.0000,   -3.0000,
           -7.0000,    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],
        [ -50.0000,  -25.0000,  -11.0000,  -19.0000,  -26.0000,   26.0000,
           70.0000,   84.0000],
        [ -21.0000,   -9.0000,   -8.0000,   -8.0000,  -14.0000,   13.0000,
           35.0000,   35.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [ -52.0000,  -38.0000,   -2.0000,  -20.0000,  -19.0000,   27.0000,
           70.0000,   84.0000],
```

```
        [ -38.0000,  -43.0000,    7.0000,  -15.0000,   -7.0000,   23.0000,
           56.0000,   56.0000],
        [  -2.0000,   -6.0000,   -3.0000,   -1.0000,    7.0000,    2.0000,
            3.0000,    7.0000],
        [  -9.0000,  -19.0000,   -4.0000,   -4.0000,   14.0000,    7.0000,
           13.0000,   21.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [ -76.0000,  -74.0000,    9.0000,  -30.0000,   54.0000,   48.0000,
           73.0000,  160.0000],
        [ -86.0000, -126.0000,    8.0000,  -35.0000,   66.0000,   47.0000,
           84.0000,  167.0000],
        [ -66.0000, -119.0000,  -14.0000,  -27.0000,   63.0000,   32.0000,
           52.0000,  130.0000],
        [ -48.0000, -106.0000,  -28.0000,  -20.0000,   54.0000,   32.0000,
           50.0000,   94.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000],
        [   0.0000,    0.0000,    0.0000,    0.0000,    0.0000,    0.0000,
            0.0000,    0.0000]], device='cuda:0', grad_fn=<TransposeBackward0>)
```

[126]:

```
### Complete this cell ###
ic_tile_id = 0
oc_tile_id = 0


kij = 0
nij = 0

bit_precision = 16
file = open('out.txt', 'w') #write to file
file.write('#out7feature0[msb-lsb],out6feature0[msb-lst],....
 ↪,out0feature0[msb-lst]#\n')
file.write('#out7feature1[msb-lsb],out6feature1[msb-lst],....
 ↪,out0feature1[msb-lst]#\n')
```

```
file.write('#................#\n')

for i in range(out.size(1)):  # time step
    for j in range(out.size(0)): # row #
        if(out[7-j,i].item() >= 0):
            out_bin = '{0:016b}'.format(round(out[7-j,i].item()))
        else:
            out_bin = '{0:016b}'.format(round(out[7-j,i].item())+65536)
        for k in range(bit_precision):
            file.write(out_bin[k])
    file.write('\n')
file.close() #close file
```

[127]:
```
print(out)
```

```
tensor([[  0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
           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,  51.0000,  13.0000,   0.0000,  37.0000,  92.0000,  51.0000,
           0.0000,  91.0000,  71.0000,  12.0000,   0.0000,  31.0000,   9.0000,
           3.0000,  43.0000],
        [  0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
           0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
           0.0000,   0.0000],
        [ 20.0000,  35.0000,   0.0000,   0.0000,  96.0000, 140.0000,  73.0000,
          34.0000, 238.0000, 370.0000, 371.0000, 242.0000,  59.0000, 201.0000,
         160.0000, 143.0000],
        [ 18.0000,  81.0000,  30.0000,   9.0000,   0.0000, 112.0000,  68.0000,
          10.0000,   0.0000, 124.0000, 119.0000,   0.0000,   0.0000,  10.0000,
          29.0000,  33.0000],
        [140.0000, 159.0000,  61.0000,  14.0000, 184.0000, 238.0000,  83.0000,
           0.0000, 230.0000, 338.0000, 200.0000,   2.0000, 256.0000, 357.0000,
         277.0000, 105.0000],
        [  9.0000,   0.0000,   0.0000,  14.0000,  22.0000,   0.0000,   0.0000,
          17.0000, 219.0000, 118.0000,  40.0000,   0.0000, 119.0000,  60.0000,
           0.0000,  88.0000]], device='cuda:0', grad_fn=<ReluBackward0>)
```

[128]:
```
### Complete this cell ###
ic_tile_id = 0
oc_tile_id = 0


kij = 0
nij = 0
```

```
relu = model.features[28]


bit_precision = 16
file = open('out_relu.txt', 'w') #write to file
file.write('#out7feature0[msb-lsb],out6feature0[msb-lst],....
 ↪,out0feature0[msb-lst]#\n')
file.write('#out7feature1[msb-lsb],out6feature1[msb-lst],....
 ↪,out0feature1[msb-lst]#\n')
file.write('#...............#\n')

for i in range(out.size(1)):  # time step
    for j in range(out.size(0)): # row #
        out_bin = relu(torch.tensor(out[7-j,i].item()))
        out_bin_1 = '{0:016b}'.format(round(out_bin.item()))
        for k in range(bit_precision):
            file.write(out_bin_1[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close() #close file
```

[129]:
```
### Complete this cell ###
ic_tile_id = 0
oc_tile_id = 0


kij = 0
nij = 0
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('out_int_relu.txt', 'w') #write to file
file.write('#out7feature0[msb-lsb],out6feature0[msb-lst],....
 ↪,out0feature0[msb-lst]#\n')
file.write('#out7feature1[msb-lsb],out6feature1[msb-lst],....
 ↪,out0feature1[msb-lst]#\n')
file.write('#...............#\n')


for i in range(out.size(1)):  # time step
    for j in range(out.size(0)): # row #
        out_bin = relu(torch.tensor(out[7-j,i].item()))
        file.write(f'{round(out_bin.item())},')
        file.write(' ')  # for visibility with blank between words, you can use␣
 ↪
    file.write('\n')
```

```
file.close() #close file
```

[105]:
```
out1 = out
print(out1[:,0])

out2 = relu(out1)
print(out2[:,0])
```

```
tensor([-114.0000,  -40.0000,  -19.0000,  -64.0000,   20.0000,   18.0000,
         140.0000,    9.0000], device='cuda:0', grad_fn=<SelectBackward0>)
tensor([  0.0000,   0.0000,   0.0000,   0.0000,  20.0000,  18.0000, 140.0000,
          9.0000], device='cuda:0', grad_fn=<SelectBackward0>)
```

[106]:
```
out
```

[106]:
```
tensor([[  0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
             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,  51.0000,  13.0000,   0.0000,  37.0000,  92.0000,  51.0000,
             0.0000,  91.0000,  71.0000,  12.0000,   0.0000,  31.0000,   9.0000,
             3.0000,  43.0000],
          [  0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
             0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
             0.0000,   0.0000],
          [ 20.0000,  35.0000,   0.0000,   0.0000,  96.0000, 140.0000,  73.0000,
            34.0000, 238.0000, 370.0000, 371.0000, 242.0000,  59.0000, 201.0000,
           160.0000, 143.0000],
          [ 18.0000,  81.0000,  30.0000,   9.0000,   0.0000, 112.0000,  68.0000,
            10.0000,   0.0000, 124.0000, 119.0000,   0.0000,   0.0000,  10.0000,
            29.0000,  33.0000],
          [140.0000, 159.0000,  61.0000,  14.0000, 184.0000, 238.0000,  83.0000,
             0.0000, 230.0000, 338.0000, 200.0000,   2.0000, 256.0000, 357.0000,
           277.0000, 105.0000],
          [  9.0000,   0.0000,   0.0000,  14.0000,  22.0000,   0.0000,   0.0000,
            17.0000, 219.0000, 118.0000,  40.0000,   0.0000, 119.0000,  60.0000,
             0.0000,  88.0000]], device='cuda:0', grad_fn=<ReluBackward0>)
```

[ ]:

[ ]: