[HW6_prob1]_VGG16_Quantization_aware_train_with_pruning

November 17, 2021

[3]:
```
import argparse
import os
import time
import shutil

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

#from tensorboardX import SummaryWriter

import torchvision
import torchvision.transforms as transforms

from models import *

import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0"

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)
```

```python
        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
```

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

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

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

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


def accuracy(output, target, topk=(1,)):
    """Computes the precision@k for the specified values of k"""
    maxk = max(topk)
    batch_size = target.size(0)

    _, pred = output.topk(maxk, 1, True, True)
    pred = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))

    res = []
    for k in topk:
        correct_k = correct[:k].view(-1).float().sum(0)
        res.append(correct_k.mul_(100.0 / batch_size))
    return res


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

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

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


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


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

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

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

```
ceil_mode=False)
    (7): QuantConv2d(
      64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): QuantConv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (14): QuantConv2d(
      128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): QuantConv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): QuantConv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): QuantConv2d(
      256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): QuantConv2d(
```

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

[4]:

```
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: 9113/10000 (91%)
```

[7]:
```
#### Prune all the QuantConv2D layers' 90% weights with 1) unstructured, and 2)␣
 ↪structured manner.
import torch.nn.utils.prune as prune

#unstructured
for layer in model.modules():
    if isinstance(layer, QuantConv2d):
        prune.l1_unstructured(layer, name='weight', amount=0.9)

#structured
'''
for layer in model.modules():
    if isinstance(layer, QuantConv2d):
        prune.ln_structured(layer, name='weight', amount=0.9, n=1)
```

```
'''
```

[7]:

```
"\nfor layer in model.modules():\n    if isinstance(layer, QuantConv2d):\n
prune.ln_structured(layer, name='weight', amount=0.9, n=1)\n"
```

[8]:

```
print(list(model.features[40].named_parameters())) # check whether there is␣
↪mask, weight_org, ...
print(model.features[40].weight) # check whether there are many zeros
```

```
[('act_alpha', Parameter containing:
tensor(1.0828, device='cuda:0', requires_grad=True)), ('weight_q', Parameter
containing:
tensor([[[[-0.0000, -0.5514, -0.0000],
          [-0.2757, -1.3786, -0.0000],
          [ 0.2757, -0.8272,  0.2757]],

         [[ 0.2757,  0.5514,  0.2757],
          [ 0.2757, -0.5514,  0.5514],
          [ 0.2757,  0.2757,  0.2757]],

         [[ 0.0000,  0.2757,  0.2757],
          [ 0.0000,  1.9301,  0.5514],
          [ 1.3786,  1.9301, -0.5514]],

         …,

         [[ 0.2757,  0.5514, -0.0000],
          [ 0.0000,  1.1029,  0.0000],
          [-0.5514, -0.0000,  0.2757]],

         [[ 0.5514,  0.5514,  0.2757],
          [ 0.8272, -0.0000,  0.2757],
          [ 0.2757, -0.0000,  0.2757]],

         [[ 0.5514, -0.2757,  0.0000],
          [ 0.5514,  1.9301, -0.2757],
          [-0.2757, -1.9301,  0.2757]]],


        [[[-0.2757,  0.5514,  0.2757],
          [-0.2757,  0.0000,  0.5514],
          [-0.2757,  0.5514,  0.0000]],

         [[-0.2757,  0.2757, -0.2757],
          [ 0.2757,  0.5514,  0.2757],
          [-0.2757,  0.0000, -0.2757]],
```

```
 [[-0.2757, -0.0000, -0.5514],
  [-0.2757, -0.0000, -0.5514],
  [-0.2757, -0.0000, -0.2757]],

 …,

 [[-0.5514, -0.2757, -0.0000],
  [ 0.2757,  0.8272,  0.0000],
  [-0.2757,  0.0000,  0.0000]],

 [[ 0.0000,  0.2757, -0.0000],
  [ 0.5514,  0.2757, -0.0000],
  [-0.2757, -0.5514, -0.0000]],

 [[ 0.2757,  0.0000,  0.2757],
  [ 0.2757,  0.2757, -0.0000],
  [ 0.0000, -0.2757, -0.2757]]],


[[[ 0.2757, -0.2757, -0.0000],
  [-0.2757,  0.2757, -0.5514],
  [-0.2757, -0.0000,  0.5514]],

 [[-0.2757, -0.0000, -0.8272],
  [-0.5514, -1.9301, -1.3786],
  [-0.2757, -0.0000,  0.2757]],

 [[-0.5514, -0.5514, -0.0000],
  [-0.2757, -0.5514,  0.8272],
  [-0.2757, -0.8272,  1.9301]],

 …,

 [[-0.5514, -0.5514,  0.5514],
  [-0.2757, -1.9301,  1.9301],
  [-0.2757,  1.9301,  1.3786]],

 [[ 0.2757, -0.0000,  0.0000],
  [-0.8272, -0.2757, -1.3786],
  [-1.1029, -0.8272,  0.5514]],

 [[ 0.2757, -0.5514, -0.5514],
  [ 0.8272, -1.1029, -1.9301],
  [-0.2757,  0.2757, -0.2757]]],


…,
```

```
[[[-0.0000,  0.0000, -0.2757],
  [-0.2757,  0.0000, -0.0000],
  [-0.2757, -0.2757,  0.0000]],

 [[ 0.0000, -0.2757, -0.0000],
  [-0.2757, -0.2757, -0.8272],
  [-0.2757, -0.0000,  0.2757]],

 [[ 0.2757, -0.2757,  0.2757],
  [-0.2757, -0.8272, -0.2757],
  [-0.8272, -0.2757,  1.9301]],

 …,

 [[-0.5514, -0.2757, -0.2757],
  [-1.3786, -1.3786,  1.1029],
  [-0.2757,  1.9301,  0.2757]],

 [[-0.2757,  0.2757,  0.5514],
  [ 0.5514,  0.2757,  1.6543],
  [-0.8272,  0.8272, -0.5514]],

 [[ 0.0000, -0.5514, -0.5514],
  [ 0.5514, -1.3786, -0.8272],
  [-0.2757,  0.8272,  1.9301]]],


[[[ 0.2757, -0.2757,  0.2757],
  [-0.2757, -0.8272, -0.5514],
  [ 0.2757, -0.5514,  0.2757]],

 [[ 0.2757, -0.2757,  1.9301],
  [ 0.0000,  0.5514,  0.2757],
  [-0.2757,  0.2757,  0.2757]],

 [[-0.0000, -0.2757, -0.2757],
  [ 0.2757, -0.5514,  1.1029],
  [-0.2757, -0.8272,  0.2757]],

 …,

 [[-0.0000, -0.0000,  1.3786],
  [ 0.0000, -0.2757, -0.5514],
  [ 0.0000, -0.8272, -0.5514]],

 [[-0.5514,  0.5514,  0.0000],
  [ 0.2757,  0.0000, -0.5514],
```

```
          [-0.2757,  0.8272, -0.2757]],

         [[-0.2757,  0.0000,  0.5514],
          [ 0.8272, -0.2757,  0.5514],
          [ 0.2757,  0.0000, -0.0000]]],


        [[[ 0.2757, -0.2757, -0.0000],
          [-0.5514,  0.5514,  0.0000],
          [ 0.5514, -0.0000, -0.0000]],

         [[ 0.5514,  1.1029, -0.2757],
          [-0.0000, -0.2757, -0.2757],
          [-0.0000, -0.0000,  0.0000]],

         [[-0.2757,  1.9301, -0.2757],
          [ 0.5514,  1.9301, -0.5514],
          [-0.2757,  0.0000, -0.0000]],

         …,

         [[ 1.9301,  0.8272,  0.0000],
          [-0.2757, -0.5514,  0.2757],
          [-0.8272, -0.2757,  0.2757]],

         [[ 1.3786,  0.5514, -0.2757],
          [-0.2757, -0.5514,  0.0000],
          [-0.5514,  0.0000, -0.0000]],

         [[-0.0000,  1.1029, -0.5514],
          [ 0.5514, -1.9301, -1.3786],
          [-0.0000, -0.2757,  0.2757]]]], device='cuda:0', requires_grad=True)),
('weight_orig', Parameter containing:
tensor([[[[ 1.2402e-04, -1.9796e-03,  7.0326e-05],
          [-4.8390e-04, -6.5048e-03, -2.0400e-04],
          [ 1.9526e-03, -2.9002e-03,  1.9334e-03]],

         [[ 1.4960e-03,  3.1367e-03,  1.7153e-03],
          [ 1.2244e-03, -1.6275e-03,  2.9924e-03],
          [ 1.5680e-03,  1.2202e-03,  1.5352e-03]],

         [[ 3.1437e-04,  1.0166e-03,  9.9755e-04],
          [ 3.3466e-04,  1.4761e-02,  2.8693e-03],
          [ 6.5087e-03,  1.5952e-02, -2.1182e-03]],

         …,

         [[ 9.0693e-04,  2.8055e-03, -3.4787e-05],
```

```
  [ 7.1053e-04,  5.5519e-03,  6.7658e-04],
  [-2.6252e-03, -1.2159e-04,  1.3696e-03]],

 [[ 2.6256e-03,  3.3385e-03,  1.2007e-03],
  [ 4.0698e-03, -8.5852e-05,  1.7846e-03],
  [ 1.5088e-03,  2.0852e-04,  1.5881e-03]],

 [[ 2.2645e-03, -1.0575e-03,  6.6150e-04],
  [ 2.3719e-03,  9.6025e-03, -3.4249e-04],
  [-9.3780e-04, -8.2512e-03,  1.2844e-03]]],


[[[-7.3995e-04,  2.2063e-03,  1.3961e-03],
  [-3.8872e-04,  7.8317e-04,  3.0451e-03],
  [-9.6147e-04,  3.1043e-03,  6.2473e-04]],

 [[-3.4335e-04,  1.3990e-03, -5.6145e-04],
  [ 1.9034e-03,  2.3212e-03,  9.9721e-04],
  [-5.5135e-04,  5.6644e-04, -9.1077e-04]],

 [[-1.2777e-03, -2.3443e-04, -1.8313e-03],
  [-4.8466e-04, -1.1972e-04, -1.8740e-03],
  [-1.1815e-03, -2.7937e-04, -5.4720e-04]],

 …,

 [[-1.6276e-03, -1.1177e-03,  1.2248e-04],
  [ 1.0702e-03,  3.5674e-03,  6.2228e-04],
  [-8.5752e-04,  6.9740e-04,  4.5008e-04]],

 [[ 7.2632e-04,  1.4405e-03,  9.3226e-06],
  [ 3.1038e-03,  1.0306e-03,  2.6920e-04],
  [-9.2305e-04, -2.3078e-03,  2.0105e-04]],

 [[ 1.9436e-03,  8.3671e-04,  1.2558e-03],
  [ 9.0271e-04,  1.7483e-03, -3.0947e-04],
  [ 6.3001e-04, -7.8490e-04, -1.5259e-03]]],


[[[ 1.2977e-03, -1.4101e-03,  1.7702e-04],
  [-1.0727e-03,  1.6777e-03, -1.7928e-03]],
  [-8.9696e-04,  2.3751e-04,  2.2932e-03]],

 [[-5.7848e-04, -3.2808e-04, -3.2820e-03],
  [-2.7216e-03, -8.0129e-03, -5.7722e-03],
  [-4.9294e-04,  2.3098e-04,  1.2814e-03]],

 [[-1.5980e-03, -1.8443e-03,  1.9139e-04],
```

```
  [-9.5728e-04, -1.8222e-03,  4.0783e-03],
  [-1.4821e-03, -3.8810e-03,  1.7675e-02]],

 …,

 [[-2.0146e-03, -2.7173e-03,  2.8608e-03],
  [-1.3281e-03, -1.8429e-02,  1.5040e-02],
  [-8.2641e-04,  1.3755e-02,  6.6097e-03]],

 [[ 1.0309e-03, -8.6632e-05,  8.0707e-04],
  [-3.1246e-03, -1.3885e-03, -5.7478e-03],
  [-4.1667e-03, -3.1951e-03,  3.1064e-03]],

 [[ 1.4363e-03, -2.5219e-03, -2.6959e-03],
  [ 3.7010e-03, -4.1965e-03, -8.1683e-03],
  [-1.3011e-03,  9.8526e-04, -1.1184e-03]]],


…,


[[[ 2.2873e-05,  6.9940e-04, -1.1692e-03],
  [-9.7597e-04,  4.2446e-04,  8.9576e-05],
  [-3.6687e-04, -1.1711e-03,  3.9825e-04]],

 [[ 4.6935e-04, -1.3452e-03, -1.7161e-04],
  [-6.6728e-04, -7.7829e-04, -3.2591e-03],
  [-1.0736e-03, -1.1455e-04,  1.3427e-03]],

 [[ 1.6876e-03, -9.9688e-04,  1.6755e-03],
  [-1.4352e-03, -3.2817e-03, -6.1526e-04],
  [-2.9136e-03, -1.4178e-03,  9.1417e-03]],

 …,

 [[-2.2097e-03, -1.4845e-03, -5.9298e-04],
  [-5.5161e-03, -6.1965e-03,  4.9957e-03],
  [-9.8764e-04,  1.1507e-02,  1.4355e-03]],

 [[-6.3502e-04,  1.8692e-03,  2.5048e-03],
  [ 2.6814e-03,  1.4512e-03,  7.7928e-03],
  [-2.9062e-03,  3.8062e-03, -1.6046e-03]],

 [[ 6.2615e-04, -2.4561e-03, -2.6417e-03],
  [ 2.7362e-03, -5.5960e-03, -3.2295e-03],
  [-1.1337e-03,  4.5225e-03,  1.4483e-02]]],
```

```
[[[ 1.7731e-03, -1.1334e-03,  1.0650e-03],
  [-1.3447e-03, -3.5666e-03, -1.6345e-03],
  [ 1.4967e-03, -1.9499e-03,  1.2605e-03]],

 [[ 1.1854e-03, -5.6255e-04,  1.0775e-02],
  [ 7.9884e-04,  2.5916e-03,  1.5235e-03],
  [-1.4411e-03,  1.5343e-03,  9.5904e-04]],

 [[ 1.4264e-04, -1.2437e-03, -1.3297e-03],
  [ 1.1714e-03, -2.6373e-03,  5.1268e-03],
  [-7.3196e-04, -3.2986e-03,  1.2197e-03]],

 …,

 [[-3.7310e-05,  4.8267e-05,  6.1050e-03],
  [ 2.8472e-04, -7.7994e-04, -2.4113e-03],
  [ 5.0692e-04, -3.7555e-03, -2.7035e-03]],

 [[-2.0473e-03,  2.2205e-03,  3.5266e-04],
  [ 1.6154e-03,  7.0074e-04, -1.9116e-03],
  [-1.5418e-03,  3.4176e-03, -1.0968e-03]],

 [[-5.3591e-04,  8.8158e-04,  3.1189e-03],
  [ 3.4673e-03, -3.7797e-04,  3.3101e-03],
  [ 1.0497e-03,  7.1264e-04,  1.9872e-04]]],


[[[ 1.6662e-03, -5.4128e-04,  8.4446e-05],
  [-2.7612e-03,  2.5475e-03,  9.0154e-04],
  [ 2.3797e-03,  2.6654e-04,  1.9204e-04]],

 [[ 2.5696e-03,  4.6747e-03, -1.1357e-03],
  [ 2.6386e-04, -6.3367e-04, -1.3375e-03],
  [-8.7767e-05, -1.9450e-04,  5.9704e-04]],

 [[-5.2930e-04,  1.9540e-02, -1.1015e-03],
  [ 2.5488e-03,  1.2713e-02, -2.4324e-03],
  [-9.0271e-04,  8.0680e-04,  2.6181e-04]],

 …,

 [[ 9.0893e-03,  3.8172e-03,  4.8077e-04],
  [-1.0006e-03, -2.2869e-03,  1.8101e-03],
  [-2.8597e-03, -4.7767e-04,  1.1425e-03]],

 [[ 6.7214e-03,  3.0569e-03, -1.0664e-03],
  [-7.8534e-04, -2.1396e-03,  7.3862e-04],
  [-1.5976e-03,  4.3934e-04, -1.5148e-05]],
```

```
         [[ 1.3486e-04,  5.7647e-03, -1.8644e-03],
          [ 2.2143e-03, -7.8414e-03, -6.4412e-03],
          [-2.2063e-04, -1.4825e-03,  1.6291e-03]]]], device='cuda:0',
       requires_grad=True)), ('weight_quant.wgt_alpha', Parameter containing:
tensor(1.9301, device='cuda:0', requires_grad=True))]
tensor([[[[ 0.0000, -0.0000,  0.0000],
          [-0.0000, -0.0065, -0.0000],
          [ 0.0000, -0.0000,  0.0000]],

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

         [[ 0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0148,  0.0000],
          [ 0.0065,  0.0160, -0.0000]],

         …,

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

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

         [[ 0.0000, -0.0000,  0.0000],
          [ 0.0000,  0.0096, -0.0000],
          [-0.0000, -0.0083,  0.0000]]],


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

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

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

         …,

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

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

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

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


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

 [[-0.0000, -0.0000, -0.0000],
  [-0.0000, -0.0080, -0.0000],
  [-0.0000,  0.0000,  0.0000]],

 [[-0.0000, -0.0000,  0.0000],
  [-0.0000, -0.0000,  0.0000],
  [-0.0000, -0.0000,  0.0177]],

 …,

 [[-0.0000, -0.0000,  0.0000],
  [-0.0000, -0.0184,  0.0150],
  [-0.0000,  0.0138,  0.0066]],

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

 [[ 0.0000, -0.0000, -0.0000],
  [ 0.0000, -0.0000, -0.0082],
  [-0.0000,  0.0000, -0.0000]]],


…,


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

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

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

[[ 0.0000, -0.0000,  0.0000],
 [-0.0000, -0.0000, -0.0000],
 [-0.0000, -0.0000,  0.0091]],

…,

[[-0.0000, -0.0000, -0.0000],
 [-0.0000, -0.0000,  0.0000],
 [-0.0000,  0.0115,  0.0000]],

[[-0.0000,  0.0000,  0.0000],
 [ 0.0000,  0.0000,  0.0078],
 [-0.0000,  0.0000, -0.0000]],

[[ 0.0000, -0.0000, -0.0000],
 [ 0.0000, -0.0000, -0.0000],
 [-0.0000,  0.0000,  0.0145]]],


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

 [[ 0.0000, -0.0000,  0.0108],
  [ 0.0000,  0.0000,  0.0000],
  [-0.0000,  0.0000,  0.0000]],

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

 …,

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

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

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

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

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

 [[-0.0000,  0.0195, -0.0000],
  [ 0.0000,  0.0127, -0.0000],
  [-0.0000,  0.0000,  0.0000]],

 …,

 [[ 0.0091,  0.0000,  0.0000],
  [-0.0000, -0.0000,  0.0000],
  [-0.0000, -0.0000,  0.0000]],

 [[ 0.0067,  0.0000, -0.0000],
  [-0.0000, -0.0000,  0.0000],
  [-0.0000,  0.0000, -0.0000]],

 [[ 0.0000,  0.0000, -0.0000],
  [ 0.0000, -0.0078, -0.0064],
  [-0.0000, -0.0000,  0.0000]]]], device='cuda:0',
grad_fn=<MulBackward0>)
```

[9]:

```
### Check sparsity ###
mask1 = model.features[40].weight_mask
sparsity_mask1 = (mask1 == 0).sum() / mask1.nelement()

print("Sparsity level: ", sparsity_mask1)
```

```
Sparsity level:  tensor(0.9000, device='cuda:0')
```

[10]:

```
## check accuracy after pruning

model.cuda()
model.eval()

test_loss = 0
correct = 0

with torch.no_grad():
    for data, target in testloader:
        data, target = data.to(device), target.to(device) # loading to GPU
        output = model(data)
```

```
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(testloader.dataset)

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

```
Test set: Accuracy: 1000/10000 (10%)
```

[11]:

```
## Start finetuning (training here), and see how much you can recover your␣
↪accuracy ##
## You can change hyper parameters such as epochs or lr ##

lr = 4e-2
weight_decay = 1e-4
epochs = 500
best_prec = 0

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

if not os.path.exists('result'):
    os.makedirs('result')
fdir = 'result/'+str(model_name)
if not os.path.exists(fdir):
    os.makedirs(fdir)


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

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

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

    # remember best precision and save checkpoint
    is_best = prec > best_prec
```

```python
    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)
```

```
Epoch: [0][0/391]       Time 0.302 (0.302)      Data 0.200 (0.200)      Loss
4.5728 (4.5728)    Prec 10.938% (10.938%)
Epoch: [0][100/391]     Time 0.063 (0.066)      Data 0.002 (0.004)      Loss
2.2522 (2.3633)    Prec 15.625% (14.380%)
Epoch: [0][200/391]     Time 0.056 (0.065)      Data 0.002 (0.003)      Loss
2.2092 (2.2550)    Prec 19.531% (16.095%)
Epoch: [0][300/391]     Time 0.061 (0.065)      Data 0.002 (0.002)      Loss
1.9751 (2.1953)    Prec 25.000% (17.097%)
Validation starts
Test: [0/79]    Time 0.191 (0.191)      Loss 2.2633 (2.2633)    Prec 17.188%
(17.188%)
 * Prec 12.490%
best acc: 12.490000
Epoch: [1][0/391]       Time 0.249 (0.249)      Data 0.200 (0.200)      Loss
2.1288 (2.1288)    Prec 15.625% (15.625%)
Epoch: [1][100/391]     Time 0.064 (0.066)      Data 0.002 (0.004)      Loss
2.3081 (2.2678)    Prec 17.969% (13.281%)
Epoch: [1][200/391]     Time 0.064 (0.065)      Data 0.002 (0.003)      Loss
2.3549 (2.2755)    Prec 11.719% (13.021%)
Epoch: [1][300/391]     Time 0.063 (0.065)      Data 0.001 (0.002)      Loss
2.2277 (2.2821)    Prec 13.281% (12.684%)
Validation starts
Test: [0/79]    Time 0.233 (0.233)      Loss 2.3296 (2.3296)    Prec 14.844%
(14.844%)
 * Prec 10.000%
best acc: 12.490000
Epoch: [2][0/391]       Time 0.255 (0.255)      Data 0.206 (0.206)      Loss
2.2868 (2.2868)    Prec 11.719% (11.719%)
Epoch: [2][100/391]     Time 0.065 (0.066)      Data 0.002 (0.004)      Loss
2.1381 (2.2548)    Prec 17.188% (13.390%)
Epoch: [2][200/391]     Time 0.067 (0.065)      Data 0.001 (0.003)      Loss
2.1856 (2.2603)    Prec 18.750% (13.211%)
Epoch: [2][300/391]     Time 0.064 (0.065)      Data 0.002 (0.002)      Loss
2.2263 (2.2324)    Prec 16.406% (14.351%)
Validation starts
Test: [0/79]    Time 0.243 (0.243)      Loss 2.3175 (2.3175)    Prec 7.812%
(7.812%)
 * Prec 10.000%
```

```
best acc: 12.490000
Epoch: [3][0/391]       Time 0.237 (0.237)      Data 0.184 (0.184)      Loss
2.2897 (2.2897)    Prec 13.281% (13.281%)
Epoch: [3][100/391]     Time 0.063 (0.066)      Data 0.002 (0.003)      Loss
1.9296 (2.1799)    Prec 21.094% (17.899%)
Epoch: [3][200/391]     Time 0.061 (0.065)      Data 0.002 (0.003)      Loss
1.8698 (2.0643)    Prec 23.438% (20.297%)
Epoch: [3][300/391]     Time 0.064 (0.065)      Data 0.002 (0.002)      Loss
1.7196 (1.9685)    Prec 31.250% (23.027%)
Validation starts
Test: [0/79]    Time 0.185 (0.185)      Loss 1.5549 (1.5549)    Prec 35.938%
(35.938%)
 * Prec 36.160%
best acc: 36.160000
Epoch: [4][0/391]       Time 0.285 (0.285)      Data 0.234 (0.234)      Loss
1.6127 (1.6127)    Prec 36.719% (36.719%)
Epoch: [4][100/391]     Time 0.064 (0.067)      Data 0.002 (0.004)      Loss
1.5699 (1.5820)    Prec 40.625% (39.372%)
Epoch: [4][200/391]     Time 0.066 (0.066)      Data 0.002 (0.003)      Loss
1.3592 (1.5127)    Prec 49.219% (42.615%)
Epoch: [4][300/391]     Time 0.063 (0.065)      Data 0.002 (0.003)      Loss
1.2980 (1.4658)    Prec 51.562% (45.102%)
Validation starts
Test: [0/79]    Time 0.171 (0.171)      Loss 1.1887 (1.1887)    Prec 57.031%
(57.031%)
 * Prec 53.220%
best acc: 53.220000
Epoch: [5][0/391]       Time 0.261 (0.261)      Data 0.210 (0.210)      Loss
1.3652 (1.3652)    Prec 53.906% (53.906%)
Epoch: [5][100/391]     Time 0.062 (0.066)      Data 0.002 (0.004)      Loss
1.2942 (1.1997)    Prec 56.250% (57.093%)
Epoch: [5][200/391]     Time 0.064 (0.065)      Data 0.002 (0.003)      Loss
1.0225 (1.1655)    Prec 66.406% (58.337%)
Epoch: [5][300/391]     Time 0.064 (0.065)      Data 0.002 (0.003)      Loss
1.0623 (1.1442)    Prec 65.625% (59.038%)
Validation starts
Test: [0/79]    Time 0.200 (0.200)      Loss 0.9787 (0.9787)    Prec 63.281%
(63.281%)
 * Prec 64.090%
best acc: 64.090000
Epoch: [6][0/391]       Time 0.225 (0.225)      Data 0.177 (0.177)      Loss
1.1043 (1.1043)    Prec 57.812% (57.812%)
Epoch: [6][100/391]     Time 0.063 (0.066)      Data 0.002 (0.004)      Loss
0.9056 (0.9731)    Prec 68.750% (65.292%)
Epoch: [6][200/391]     Time 0.064 (0.065)      Data 0.002 (0.003)      Loss
0.8470 (0.9392)    Prec 67.188% (66.608%)
Epoch: [6][300/391]     Time 0.064 (0.065)      Data 0.002 (0.002)      Loss
0.9764 (0.9145)    Prec 71.875% (67.652%)
```

```
Validation starts
Test: [0/79]    Time 0.204 (0.204)      Loss 0.7748 (0.7748)    Prec 69.531%
(69.531%)
 * Prec 70.820%
best acc: 70.820000
Epoch: [7][0/391]       Time 0.277 (0.277)      Data 0.223 (0.223)      Loss
0.9007 (0.9007)    Prec 70.312% (70.312%)
Epoch: [7][100/391]     Time 0.065 (0.066)      Data 0.002 (0.004)      Loss
0.6899 (0.7698)    Prec 76.562% (73.136%)
Epoch: [7][200/391]     Time 0.062 (0.065)      Data 0.002 (0.003)      Loss
0.7931 (0.7533)    Prec 69.531% (73.640%)
Epoch: [7][300/391]     Time 0.063 (0.065)      Data 0.002 (0.003)      Loss
0.7277 (0.7314)    Prec 77.344% (74.543%)
Validation starts
Test: [0/79]    Time 0.244 (0.244)      Loss 0.7439 (0.7439)    Prec 71.875%
(71.875%)
 * Prec 75.900%
best acc: 75.900000
Epoch: [8][0/391]       Time 0.282 (0.282)      Data 0.231 (0.231)      Loss
0.5962 (0.5962)    Prec 78.125% (78.125%)
Epoch: [8][100/391]     Time 0.064 (0.066)      Data 0.002 (0.004)      Loss
0.4643 (0.6088)    Prec 84.375% (79.038%)
Epoch: [8][200/391]     Time 0.066 (0.065)      Data 0.002 (0.003)      Loss
0.5297 (0.6111)    Prec 83.594% (78.984%)
Epoch: [8][300/391]     Time 0.064 (0.065)      Data 0.002 (0.003)      Loss
0.6117 (0.6051)    Prec 78.125% (79.272%)
Validation starts
Test: [0/79]    Time 0.206 (0.206)      Loss 0.5042 (0.5042)    Prec 85.938%
(85.938%)
 * Prec 80.860%
best acc: 80.860000
Epoch: [9][0/391]       Time 0.243 (0.243)      Data 0.194 (0.194)      Loss
0.4702 (0.4702)    Prec 85.938% (85.938%)
Epoch: [9][100/391]     Time 0.065 (0.066)      Data 0.002 (0.004)      Loss
0.4480 (0.5360)    Prec 86.719% (81.722%)
Epoch: [9][200/391]     Time 0.065 (0.065)      Data 0.002 (0.003)      Loss
0.5198 (0.5223)    Prec 80.469% (82.210%)
Epoch: [9][300/391]     Time 0.063 (0.065)      Data 0.002 (0.002)      Loss
0.3839 (0.5163)    Prec 85.938% (82.454%)
Validation starts
Test: [0/79]    Time 0.197 (0.197)      Loss 0.5566 (0.5566)    Prec 80.469%
(80.469%)
 * Prec 81.380%
best acc: 81.380000
Epoch: [10][0/391]      Time 0.249 (0.249)      Data 0.198 (0.198)      Loss
0.3747 (0.3747)    Prec 85.938% (85.938%)
Epoch: [10][100/391]    Time 0.063 (0.066)      Data 0.001 (0.004)      Loss
0.5029 (0.4611)    Prec 84.375% (84.360%)
```

```
Epoch: [10][200/391]    Time 0.057 (0.065)      Data 0.002 (0.003)      Loss
0.4235 (0.4519)    Prec 83.594% (84.674%)
Epoch: [10][300/391]    Time 0.063 (0.065)      Data 0.002 (0.002)      Loss
0.4212 (0.4506)    Prec 83.594% (84.681%)
Validation starts
Test: [0/79]    Time 0.201 (0.201)      Loss 0.4448 (0.4448)    Prec 85.156%
(85.156%)
 * Prec 82.770%
best acc: 82.770000
Epoch: [11][0/391]      Time 0.234 (0.234)      Data 0.190 (0.190)      Loss
0.4359 (0.4359)    Prec 86.719% (86.719%)
Epoch: [11][100/391]    Time 0.064 (0.066)      Data 0.002 (0.004)      Loss
0.3535 (0.3900)    Prec 88.281% (86.502%)
Epoch: [11][200/391]    Time 0.065 (0.065)      Data 0.002 (0.003)      Loss
0.3467 (0.3978)    Prec 87.500% (86.392%)
Epoch: [11][300/391]    Time 0.064 (0.065)      Data 0.002 (0.002)      Loss
0.3472 (0.4010)    Prec 89.844% (86.272%)
Validation starts
Test: [0/79]    Time 0.185 (0.185)      Loss 0.3543 (0.3543)    Prec 86.719%
(86.719%)
 * Prec 84.760%
best acc: 84.760000
Epoch: [12][0/391]      Time 0.246 (0.246)      Data 0.189 (0.189)      Loss
0.2864 (0.2864)    Prec 88.281% (88.281%)
Epoch: [12][100/391]    Time 0.064 (0.066)      Data 0.002 (0.004)      Loss
0.3805 (0.3515)    Prec 89.062% (88.304%)
Epoch: [12][200/391]    Time 0.064 (0.065)      Data 0.001 (0.003)      Loss
0.3693 (0.3527)    Prec 86.719% (88.196%)
Epoch: [12][300/391]    Time 0.061 (0.065)      Data 0.002 (0.002)      Loss
0.1897 (0.3545)    Prec 95.312% (88.009%)
Validation starts
Test: [0/79]    Time 0.194 (0.194)      Loss 0.4719 (0.4719)    Prec 84.375%
(84.375%)
 * Prec 84.830%
best acc: 84.830000
Epoch: [13][0/391]      Time 0.260 (0.260)      Data 0.203 (0.203)      Loss
0.3046 (0.3046)    Prec 90.625% (90.625%)
```

```
        ␣
 ↪---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call␣
 ↪last)

        /tmp/ipykernel_95/2932872664.py in <module>
         23     adjust_learning_rate(optimizer, epoch)
         24
```

```
---> 25     train(trainloader, model, criterion, optimizer, epoch)
     26
     27     # evaluate on test set


    /tmp/ipykernel_95/1701083248.py in train(trainloader, model, criterion,␣
↪optimizer, epoch)
     78
     79         # compute output
---> 80         output = model(input)
     81         loss = criterion(output, target)
     82


    /opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py in␣
↪_call_impl(self, *input, **kwargs)
   1049         if not (self._backward_hooks or self._forward_hooks or self.
↪_forward_pre_hooks or _global_backward_hooks
   1050                 or _global_forward_hooks or␣
↪_global_forward_pre_hooks):
-> 1051             return forward_call(*input, **kwargs)
   1052         # Do not call functions when jit is used
   1053         full_backward_hooks, non_full_backward_hooks = [], []


    ~/HW6/models/vgg_quant.py in forward(self, x)
     23
     24     def forward(self, x):
---> 25         out = self.features(x)
     26         out = out.view(out.size(0), -1)
     27         out = self.classifier(out)


    /opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py in␣
↪_call_impl(self, *input, **kwargs)
   1049         if not (self._backward_hooks or self._forward_hooks or self.
↪_forward_pre_hooks or _global_backward_hooks
   1050                 or _global_forward_hooks or␣
↪_global_forward_pre_hooks):
-> 1051             return forward_call(*input, **kwargs)
   1052         # Do not call functions when jit is used
   1053         full_backward_hooks, non_full_backward_hooks = [], []


    /opt/conda/lib/python3.9/site-packages/torch/nn/modules/container.py in␣
↪forward(self, input)
    137     def forward(self, input):
```

```
    138         for module in self:
--> 139             input = module(input)
    140         return input
    141


/opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py in
↪_call_impl(self, *input, **kwargs)
   1069             input = bw_hook.setup_input_hook(input)
   1070
-> 1071         result = forward_call(*input, **kwargs)
   1072         if _global_forward_hooks or self._forward_hooks:
   1073             for hook in itertools.chain(


~/HW6/models/quant_layer.py in forward(self, x)
    101
    102     def forward(self, x):
--> 103         weight_q = self.weight_quant(self.weight)
    104         #self.register_parameter('weight_q', Parameter(weight_q))  #
↪Mingu added
    105         self.weight_q = torch.nn.Parameter(weight_q)  # Store
↪weight_q during the training


/opt/conda/lib/python3.9/site-packages/torch/nn/modules/module.py in
↪_call_impl(self, *input, **kwargs)
   1049         if not (self._backward_hooks or self._forward_hooks or self.
↪_forward_pre_hooks or _global_backward_hooks
   1050                 or _global_forward_hooks or
↪_global_forward_pre_hooks):
-> 1051             return forward_call(*input, **kwargs)
   1052         # Do not call functions when jit is used
   1053         full_backward_hooks, non_full_backward_hooks = [], []


~/HW6/models/quant_layer.py in forward(self, weight)
     53         mean = weight.data.mean()
     54         std = weight.data.std()
---> 55         weight = weight.add(-mean).div(std)      # weights
↪normalization
     56         weight_q = self.weight_q(weight, self.wgt_alpha)
     57


KeyboardInterrupt:
```

[12]:
```
## check your accuracy again after finetuning
model.cuda()
model.eval()

test_loss = 0
correct = 0

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

test_loss /= len(testloader.dataset)

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

```
Test set: Accuracy: 8546/10000 (85%)
```

[13]:
```
## Send an image and use prehook to grab the inputs of all the QuantConv2d␣
↪layers

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
```

[14]:
```
##### Find "weight_int" for features[3] ####
w_bit = 4
weight_q = model.features[3].weight_q
w_alpha = model.features[3].weight_quant.wgt_alpha
w_delta = w_alpha /(2**(w_bit-1)-1)

weight_int = weight_q / w_delta
print(weight_int)
```

```
tensor([[[[ 0.0000,  0.0000, -2.0000],
          [-3.0000, -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, -7.0000],
          [-7.0000,  0.0000,  0.0000],
          [ 0.0000,  5.0000,  0.0000]],

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

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

 …,

 [[ 0.0000, -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, -3.0000],
  [ 0.0000,  0.0000,  0.0000]],

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

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

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

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

 …,

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

 [[ 0.0000,  5.0000,  0.0000],
  [ 0.0000,  0.0000, -5.0000],
  [ 0.0000,  0.0000,  0.0000]],

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


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

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

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

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

  …,

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

  [[ 0.0000,  0.0000,  0.0000],
   [ 0.0000,  0.0000,  0.0000],
   [ 0.0000,  0.0000,  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],
   [ 0.0000,  0.0000, -3.0000],
   [ 0.0000,  0.0000,  0.0000]],

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

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

  …,

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

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

[15]:
```python
#### check your sparsity for weight_int is near 90% #####
#### Your sparsity could be >90% after quantization #####
sparsity_weight_int = (weight_int == 0).sum() / weight_int.nelement()
print("Sparsity level: ", sparsity_weight_int)
```

```
Sparsity level:  tensor(0.9054, device='cuda:0')
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: