

[HW6_prob1]_VGG16_Quantization_aware_train_with_pruning

November 26, 2025

```
[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 torch.nn.utils.prune as prune

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)
        loss = criterion(output, target)

        # measure accuracy and record loss
        prec = accuracy(output, target)[0]
        losses.update(loss.item(), input.size(0))
        top1.update(prec.item(), input.size(0))

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

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

    if i % print_freq == 0:
        print('Epoch: [{0}][{1}/{2}]\t'
              'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
              'Data {data_time.val:.3f} ({data_time.avg:.3f})\t'
              'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
              'Prec {top1.val:.3f}% ({top1.avg:.3f}%)'.format(
                  epoch, i, len(trainloader), batch_time=batch_time,
                  data_time=data_time, loss=losses, top1=top1))

def validate(val_loader, model, criterion):
    batch_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    # switch to evaluate mode
    model.eval()

    end = time.time()
    with torch.no_grad():
        for i, (input, target) in enumerate(val_loader):

            input, target = input.cuda(), target.cuda()

            # compute output
            output = model(input)
            loss = criterion(output, target)

```

```

# measure accuracy and record loss
prec = accuracy(output, target)[0]
losses.update(loss.item(), input.size(0))
top1.update(prec.item(), input.size(0))

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

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

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

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

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

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

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

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

```

```

        self.count = 0

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

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

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

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

```

```

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

```

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

```

```

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

```

```
[2]: PATH = "result/VGG16_quant/model_best.pth.tar"
checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['state_dict'])
device = torch.device("cuda")

model.cuda()
model.eval()

test_loss = 0
correct = 0

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

test_loss /= len(testloader.dataset)

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

Test set: Accuracy: 8745/10000 (87%)

```
[13]: # 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
```

```
[3]: # Apply unstructured pruning with 80% sparsity for the weights for all the convu
      ↪layers
for layer in model.features:
    if isinstance(layer, QuantConv2d):
        prune.l1_unstructured(layer, name='weight', amount=0.8)
```

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

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

Sparsity level: tensor(0.8000, device='cuda:0')

```
[5]: ## 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}%)'.format(
    correct, len(testloader.dataset),
    100. * correct / len(testloader.dataset)))
```

Test set: Accuracy: 995/10000 (10%)

```
[8]: ## Start finetuning (training here), and see how much you can recover your ↵
      ↵accuracy ##

## You can change hyper parameters such as epochs or lr ##

model_name = "VGG16_quant_unstructured_pruning"

lr = 1e-3
weight_decay = 1e-5
epochs = 80
best_prec = 0

criterion = nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.85, ↵
    ↵weight_decay=weight_decay)

fdirectory = 'result/' + str(model_name)
checkpoint = torch.load(fdirectory + '/model_best.pth.tar')
model.load_state_dict(checkpoint['state_dict'])
model.cuda()

# 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)

```

[8]: 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)
)

```

[9]: *## 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}%)'.format(
    correct, len(testloader.dataset),
    100. * correct / len(testloader.dataset)))

```

Test set: Accuracy: 8073/10000 (81%)

[10]: *## 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 = next(dataiter)
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
```

```
[11]: ##### 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)
```

```
[12]: ##### check your sparsity for weight_int is near 80% #####  
##### Your sparsity could be >80% after quantization #####  
sparsity_weight_int = (weight_int == 0).sum() / weight_int.nelement()  
print("Sparsity level: ", sparsity_weight_int)
```

```
Sparsity level: tensor(0.8086, device='cuda:0')
```

```
[13]: model_name = "VGG16_quant"  
model = VGG16_quant()  
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}%)'.format(  
          correct, len(testloader.dataset),  
          100. * correct / len(testloader.dataset)))
```

```
Test set: Accuracy: 8745/10000 (87%)
```

```
[14]: # Apply structured pruning with 80% sparsity for the weights for all the convu
      ↵layers

for layer in model.features:
    if isinstance(layer, QuantConv2d):
        prune.ln_structured(layer, name='weight', amount=0.8, dim=1, u
      ↵n=float('-inf'))
```

```
[15]: ## 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}%)'.format(
    correct, len(testloader.dataset),
    100. * correct / len(testloader.dataset)))
```

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

```
[16]: ## Start finetuning (training here), and see how much you can recover youru
      ↵accuracy ##

## You can change hyper parameters such as epochs or lr ##

model_name = "VGG16_quant_structured_pruning"

lr = 1e-3
weight_decay = 1e-5
epochs = 50
best_prec = 0
```

```

criterion = nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.85, weight_decay=weight_decay)

fdirectory = 'result/' + str(model_name)
checkpoint = torch.load(fdirectory + '/model_best.pth.tar')
model.load_state_dict(checkpoint['state_dict'])
model.cuda()

# 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, fdirectory)

```

[16]: 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)
)

```

[17]: *### Check sparsity ###*
mask1 = model.features[40].weight_mask

```
sparsity_mask1 = (mask1 == 0).sum() / mask1.nelement()

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

Sparsity level: tensor(0.8008, device='cuda:0')

[18]: *## 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}%)'.format(
    correct, len(testloader.dataset),
    100. * correct / len(testloader.dataset)))
```

Test set: Accuracy: 7540/10000 (75%)