```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.backends.cudnn as cudnn
import torchvision
import torchvision.transforms as transforms
import os, time, shutil
from models import *
```

```
# Device Selection
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
cudnn.benchmark = True
torch.backends.cudnn.fastest = True   # maximize RTX 3080 throughput
print(f"=> Using device: {device}")
```

```
=> Using device: cuda
```

```
# Training Parameters
batch_size = 128
epochs = 200
lr = 0.1
```

```
best_acc = 0
save_dir = "result/VGG16_quant"
os.makedirs(save_dir, exist_ok=True)
```

```
model = VGG16_quant().to(device)
if torch.cuda.device_count() > 1:
    print(f"=> Using {torch.cuda.device_count()} GPUs")
    model = nn.DataParallel(model)

# Fix: Reinitialize the classifier layer with the correct input features
# Based on VGG16_quant architecture and CIFAR10 input (32x32 with 5 max-pooling layers),
# the spatial dimension becomes 1x1, and the last conv block outputs 16 channels.
num_features_after_flattening = 16
model.classifier = nn.Linear(num_features_after_flattening, 10).to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.7, weight_decay=5e-4)
#scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[25, 40], gamma=0.1)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)
scaler = torch.amp.GradScaler(device='cuda') # Updated to recommended syntax

normalize = transforms.Normalize(mean=[0.491, 0.482, 0.447],
                                 std=[0.247, 0.243, 0.262])

train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize,
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    normalize,
])

train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=train_transform)
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=test_transform)

train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=256,              # 128 → 256 (fits in 10 GB easily)
    shuffle=True,
    num_workers=os.cpu_count(),  # use all CPU cores
    pin_memory=True,
    persistent_workers=True,
    prefetch_factor=4,           # overlap data loading with compute
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=512,              # eval can use larger batch
    shuffle=False,
    num_workers=os.cpu_count(),
    pin_memory=True,
    persistent_workers=True,
)

class AverageMeter:
    def __init__(self): self.reset()
    def reset(self): self.val=self.avg=self.sum=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 accuracy(output, target, topk=(1,)):
    maxk = max(topk)
    _, 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].reshape(-1).float().sum(0)
        res.append(correct_k.mul_(100.0 / target.size(0)))
    return res

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 train(train_loader, model, criterion, optimizer, epoch):
    model.train()
    losses, top1 = AverageMeter(), AverageMeter()
    start = time.time()

    current_lr = optimizer.param_groups[0]['lr']

    for i, (inputs, targets) in enumerate(train_loader):
        inputs = inputs.to(device, non_blocking=True)
        targets = targets.to(device, non_blocking=True)

        optimizer.zero_grad(set_to_none=True)

        with torch.cuda.amp.autocast():
            outputs = model(inputs)
            loss = criterion(outputs, targets)

        # accuracy in full precision is fine (small overhead)
        prec1 = accuracy(outputs, targets)[0]
        losses.update(loss.item(), inputs.size(0))
        top1.update(prec1.item(), inputs.size(0))

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        if i % 100 == 0:
            print(
                f"Epoch [{epoch}] [{i}/{len(train_loader)}] "
                f"LR {current_lr:.5e}  "
                f"Loss {losses.val:.4f} ({losses.avg:.4f})  "
                f"Acc {top1.val:.2f}% ({top1.avg:.2f}%)"
            )

    print(
        f" Epoch {epoch} done in {time.time()-start:.1f}s | "
        f"LR: {current_lr:.5e} | Train Acc: {top1.avg:.2f}% | Loss: {losses.avg:.4f}"
    )

def validate(val_loader, model, criterion, epoch):
    model.eval()
    losses, top1 = AverageMeter(), AverageMeter()
    with torch.no_grad():
        for i, (inputs, targets) in enumerate(val_loader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            prec1 = accuracy(outputs, targets)[0]
            losses.update(loss.item(), inputs.size(0))
            top1.update(prec1.item(), inputs.size(0))
    print(f"Validation Epoch {epoch}: Acc {top1.avg:.2f}% | Loss {losses.avg:.4f}")
    return top1.avg
```

```
# Training Loop
for epoch in range(1, epochs+1):
    train(train_loader, model, criterion, optimizer, epoch)
    val_acc = validate(test_loader, model, criterion, epoch)
    scheduler.step()

    is_best = val_acc > best_acc
    best_acc = max(val_acc, best_acc)

    save_checkpoint({
        'epoch': epoch,
        'state_dict': model.state_dict(),
        'best_acc': best_acc,
        'optimizer': optimizer.state_dict(),
    }, is_best, save_dir)

    print(f"Epoch {epoch} complete | Best Acc: {best_acc:.2f}%\n")

print("Training completed. Best accuracy: {:.2f}%".format(best_acc))
```

```
/tmp/ipython-input-2022107039.py:90: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `to
  with torch.cuda.amp.autocast():
Epoch [1] [0/196] LR 1.00000e-01  Loss 2.4543 (2.4543)  Acc 12.50% (12.50%)
Epoch [1] [100/196] LR 1.00000e-01  Loss 1.9829 (2.1364)  Acc 25.39% (19.06%)
 Epoch 1 done in 57.7s | LR: 1.00000e-01 | Train Acc: 23.10% | Loss: 2.0231
Validation Epoch 1: Acc 28.97% | Loss 2.0224
Epoch 1 complete | Best Acc: 28.97%

Epoch [2] [0/196] LR 9.99938e-02  Loss 1.8712 (1.8712)  Acc 30.86% (30.86%)
Epoch [2] [100/196] LR 9.99938e-02  Loss 1.8153 (1.8319)  Acc 28.52% (30.94%)
 Epoch 2 done in 21.1s | LR: 9.99938e-02 | Train Acc: 31.95% | Loss: 1.8109
Validation Epoch 2: Acc 28.65% | Loss 2.0222
Epoch 2 complete | Best Acc: 28.97%

Epoch [3] [0/196] LR 9.99753e-02  Loss 1.8150 (1.8150)  Acc 33.20% (33.20%)
Epoch [3] [100/196] LR 9.99753e-02  Loss 1.7309 (1.7349)  Acc 41.41% (36.55%)
 Epoch 3 done in 20.3s | LR: 9.99753e-02 | Train Acc: 37.51% | Loss: 1.7128
Validation Epoch 3: Acc 34.74% | Loss 1.8825
Epoch 3 complete | Best Acc: 34.74%

Epoch [4] [0/196] LR 9.99445e-02  Loss 1.6215 (1.6215)  Acc 42.58% (42.58%)
Epoch [4] [100/196] LR 9.99445e-02  Loss 1.6810 (1.6189)  Acc 38.28% (40.88%)
 Epoch 4 done in 19.7s | LR: 9.99445e-02 | Train Acc: 42.28% | Loss: 1.5875
Validation Epoch 4: Acc 38.27% | Loss 1.7405
Epoch 4 complete | Best Acc: 38.27%

Epoch [5] [0/196] LR 9.99013e-02  Loss 1.5682 (1.5682)  Acc 47.27% (47.27%)
Epoch [5] [100/196] LR 9.99013e-02  Loss 1.4291 (1.4837)  Acc 47.27% (46.43%)
 Epoch 5 done in 20.5s | LR: 9.99013e-02 | Train Acc: 47.10% | Loss: 1.4602
Validation Epoch 5: Acc 42.21% | Loss 1.6138
Epoch 5 complete | Best Acc: 42.21%

Epoch [6] [0/196] LR 9.98459e-02  Loss 1.3770 (1.3770)  Acc 47.27% (47.27%)
Epoch [6] [100/196] LR 9.98459e-02  Loss 1.4023 (1.3767)  Acc 50.39% (50.69%)
 Epoch 6 done in 20.7s | LR: 9.98459e-02 | Train Acc: 51.62% | Loss: 1.3544
Validation Epoch 6: Acc 51.00% | Loss 1.4012
Epoch 6 complete | Best Acc: 51.00%

Epoch [7] [0/196] LR 9.97781e-02  Loss 1.4644 (1.4644)  Acc 45.70% (45.70%)
Epoch [7] [100/196] LR 9.97781e-02  Loss 1.2481 (1.2938)  Acc 59.38% (53.75%)
 Epoch 7 done in 20.8s | LR: 9.97781e-02 | Train Acc: 54.54% | Loss: 1.2731
Validation Epoch 7: Acc 54.81% | Loss 1.2984
Epoch 7 complete | Best Acc: 54.81%

Epoch [8] [0/196] LR 9.96980e-02  Loss 1.2268 (1.2268)  Acc 56.25% (56.25%)
Epoch [8] [100/196] LR 9.96980e-02  Loss 1.2938 (1.2074)  Acc 53.12% (56.80%)
 Epoch 8 done in 20.2s | LR: 9.96980e-02 | Train Acc: 57.58% | Loss: 1.1923
Validation Epoch 8: Acc 54.30% | Loss 1.2940
Epoch 8 complete | Best Acc: 54.81%

Epoch [9] [0/196] LR 9.96057e-02  Loss 1.1379 (1.1379)  Acc 64.84% (64.84%)
Epoch [9] [100/196] LR 9.96057e-02  Loss 1.0648 (1.1485)  Acc 61.33% (59.32%)
 Epoch 9 done in 19.8s | LR: 9.96057e-02 | Train Acc: 60.26% | Loss: 1.1335
Validation Epoch 9: Acc 56.24% | Loss 1.3087
Epoch 9 complete | Best Acc: 56.24%
Epoch [10] [0/196] LR 9.95012e-02  Loss 0.9137 (0.9137)  Acc 67.97% (67.97%)
```

```
# HW

#  1. Train with 4 bits for both weight and activation to achieve >90% accuracy
#  2. Find x_int and w_int for the 2nd convolution layer
#  3. Check the recovered psum has similar value to the un-quantized original psum
#     (such as example 1 in W3S2)
```

```
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 test_loader:
        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(test_loader.dataset)

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

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipython-input-1934533436.py in <cell line: 0>()
      1 PATH = "result/VGG16_quant/model_best.pth.tar"
      2 checkpoint = torch.load(PATH)
----> 3 model.load_state_dict(checkpoint['state_dict'])
      4 device = torch.device("cuda")
      5

/usr/local/lib/python3.12/dist-packages/torch/nn/modules/module.py in load_state_dict(self, state_dict, strict,
assign)
   2627
   2628         if len(error_msgs) > 0:
-> 2629             raise RuntimeError(
   2630                 "Error(s) in loading state_dict for {}:\n\t{}".format(
   2631                     self.__class__.__name__, "\n\t".join(error_msgs)

RuntimeError: Error(s) in loading state_dict for VGG_quant:
	size mismatch for features.34.weight: copying a param with shape torch.Size([16, 512, 3, 3]) from checkpoint,
the shape in current model is torch.Size([512, 512, 3, 3]).
	size mismatch for features.34.weight_q: copying a param with shape torch.Size([16, 512, 3, 3]) from checkpoint,
the shape in current model is torch.Size([512, 512, 3, 3]).
	size mismatch for features.35.weight: copying a param with shape torch.Size([16]) from checkpoint, the shape in
current model is torch.Size([512]).
	size mismatch for features.35.bias: copying a param with shape torch.Size([16]) from checkpoint, the shape in
current model is torch.Size([512]).
	size mismatch for features.35.running_mean: copying a param with shape torch.Size([16]) from checkpoint, the
shape in current model is torch.Size([512]).
	size mismatch for features.35.running_var: copying a param with shape torch.Size([16]) from checkpoint, the
shape in current model is torch.Size([512]).
	size mismatch for features.37.weight: copying a param with shape torch.Size([16, 16, 3, 3]) from checkpoint,
the shape in current model is torch.Size([512, 512, 3, 3]).
	size mismatch for features.37.weight_q: copying a param with shape torch.Size([16, 16, 3, 3]) from checkpoint,
the shape in current model is torch.Size([512, 512, 3, 3]).
	size mismatch for features.38.weight: copying a param with shape torch.Size([16]) from checkpoint, the shape in
current model is torch.Size([512]).
	size mismatch for features.38.bias: copying a param with shape torch.Size([16]) from checkpoint, the shape in
current model is torch.Size([512]).
	size mismatch for features.38.running_mean: copying a param with shape torch.Size([16]) from checkpoint, the
shape in current model is torch.Size([512]).
	size mismatch for features.38.running_var: copying a param with shape torch.Size([16]) from checkpoint, the
shape in current model is torch.Size([512]).
	size mismatch for features.40.weight: copying a param with shape torch.Size([16, 16, 3, 3]) from checkpoint,
the shape in current model is torch.Size([512, 512, 3, 3]).
	size mismatch for features.40.weight_q: copying a param with shape torch.Size([16, 16, 3, 3]) from checkpoint,
the shape in current model is torch.Size([512, 512, 3, 3]).
	size mismatch for features.41.weight: copying a param with shape torch.Size([16]) from checkpoint, the shape in
current model is torch.Size([512]).
	size mismatch for features.41.bias: copying a param with shape torch.Size([16]) from checkpoint, the shape in
current model is torch.Size([512]).
	size mismatch for features.41.running_mean: copying a param with shape torch.Size([16]) from checkpoint, the
shape in current model is torch.Size([512]).
	size mismatch for features.41.running_var: copying a param with shape torch.Size([16]) from checkpoint, the
shape in current model is torch.Size([512]).
```

```python
PATH = "result/VGG16_quant/model_best.pth.tar"
checkpoint = torch.load(PATH)

print("Keys in the state_dict from model_best.pth.tar:")
for key, value in checkpoint['state_dict'].items():
    print(f"  {key}: {value.shape}")
```

```python
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(test_loader)
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
```

```
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)
```

```
act = save_output.outputs[1][0]   # check this
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([[[[ 0.0000,  0.0000,  0.0000,  ...,  9.0000,  9.0000,  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],
          ...,
          [ 2.0000,  8.0000, 12.0000,  ...,  9.0000, 10.0000,  7.0000],
          [ 8.0000,  9.0000, 12.0000,  ...,  9.0000,  7.0000,  5.0000],
          [ 0.0000,  0.0000,  2.0000,  ...,  3.0000,  0.0000,  0.0000]],

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

         [[ 3.0000,  5.0000,  5.0000,  ...,  8.0000,  8.0000,  8.0000],
          [ 0.0000,  2.0000,  5.0000,  ...,  2.0000,  2.0000,  4.0000],
          [ 1.0000,  3.0000,  3.0000,  ...,  2.0000,  2.0000,  4.0000],
          ...,
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  3.0000],
          [ 1.0000,  0.0000,  1.0000,  ...,  0.0000,  0.0000,  2.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  1.0000]],

         ...,

         [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  3.0000,  1.0000,  ...,  6.0000,  6.0000,  6.0000],
          [ 0.0000,  2.0000,  3.0000,  ...,  6.0000,  6.0000,  6.0000],
          ...,
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

         [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          ...,
          [ 6.0000,  8.0000,  8.0000,  ...,  7.0000,  6.0000,  3.0000],
          [ 7.0000,  9.0000,  6.0000,  ...,  5.0000,  5.0000,  3.0000],
          [ 3.0000,  5.0000,  5.0000,  ...,  6.0000,  6.0000,  5.0000]],
         [[ 7.0000,  7.0000,  3.0000,  ...,  5.0000,  5.0000,  0.0000],
          [ 0.0000,  2.0000,  1.0000,  ...,  0.0000,  0.0000,  1.0000],
          [ 0.0000,  0.0000,  2.0000,  ...,  0.0000,  0.0000,  1.0000],
          ...,
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  4.0000,  1.0000],
          [ 0.0000,  2.0000,  2.0000,  ...,  0.0000,  1.0000,  2.0000],
          [ 9.0000, 15.0000, 15.0000,  ..., 15.0000, 15.0000, 15.0000]]],


        [[[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          ...,
          [12.0000, 15.0000, 10.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 9.0000,  7.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
          [ 3.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  2.0000]],
```

```
# Changed the code to mach the dimensions
conv_int = torch.nn.Conv2d(in_channels = 16, out_channels=16, kernel_size = 3, padding=1)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
conv_int.bias = model.features[3].bias
output_int = conv_int(act_int)
output_recovered = output_int * (act_alpha / (2**act_bit-1)) * (w_alpha / (2**(w_bit-1)-1))
print(output_recovered)
```

```
tensor([[[[-1.8148e+00, -6.0575e+00, -4.2911e+00,  ..., -4.3314e+00,
           -7.4448e+00, -1.1131e+00],
          [-3.0247e+00, -6.5173e+00, -4.2669e+00,  ..., -5.4687e+00,
           -5.2993e+00, -1.0002e+00],
          [-5.3235e+00, -2.5004e+00, -5.8720e+00,  ..., -4.9202e+00,
           -5.0977e+00, -2.3875e+00],
          ...,
          [-1.0445e+01, -1.0203e+01, -1.0566e+01,  ..., -1.4882e+01,
            3.4038e+00, -1.4382e+01],
          [-1.3970e+01, -7.5336e+00, -1.1187e+01,  ..., -5.4042e-01,
           -6.8641e+00, -9.5500e+00],
          [-5.8478e+00, -5.2025e+00, -4.0410e+00,  ...,  4.3637e+00,
           -2.2988e+00,  3.2909e+00]],

         [[ 7.1222e+00,  1.2712e+01,  9.3081e+00,  ...,  9.9856e+00,
            8.7999e+00,  4.7831e+00],
          [ 1.2494e+01,  9.4210e+00,  7.7675e+00,  ...,  5.4848e+00,
            3.3716e+00, -2.5004e-01],
          [ 8.7193e+00,  7.7513e+00,  6.7108e+00,  ...,  6.4124e+00,
            3.4522e+00,  1.9278e+00],
          ...,
          [ 7.6384e+00,  6.8722e+00,  5.6784e+00,  ...,  7.1867e+00,
            5.1461e+00,  2.8795e+00],
          [ 5.4122e+00,  5.8317e+00,  8.4127e+00,  ...,  7.0657e+00,
            5.3074e+00, -3.9523e-01],
          [ 2.2827e+00,  4.8073e+00,  3.7910e+00,  ..., -1.4035e+00,
           -3.9362e+00, -1.9358e+00]],

         [[ 9.0016e+00,  1.5019e+01,  1.3857e+01,  ...,  1.6011e+01,
            1.6382e+01,  1.4212e+01],
          [ 9.3968e+00,  1.3825e+01,  1.2470e+01,  ...,  1.1171e+01,
            1.3607e+01,  1.2462e+01],
          [ 7.6062e+00,  1.0728e+01,  8.1224e+00,  ...,  9.1951e+00,
            1.1341e+01,  1.1341e+01],
          ...,
          [ 1.5785e+01,  2.0778e+01,  2.6085e+01,  ...,  2.4520e+01,
            2.6303e+01,  1.8898e+01],
          [ 1.6164e+01,  2.3327e+01,  2.7811e+01,  ...,  2.6609e+01,
            2.9763e+01,  2.2722e+01],
          [ 1.5785e+01,  2.1286e+01,  2.3617e+01,  ...,  2.2988e+01,
            2.7037e+01,  1.8447e+01]],

         ...,

         [[-1.1583e+01, -1.1462e+01, -1.1663e+01,  ..., -6.7673e+00,
           -1.6454e+00,  2.6295e+00],
          [-1.1091e+01, -9.3968e+00, -1.3309e+01,  ..., -7.6223e+00,
           -5.6461e-01, -5.8881e-01],
          [-1.0171e+01, -9.5904e+00, -1.1462e+01,  ..., -5.8317e+00,
            3.6297e-01, -7.8239e-01],
          ...,
          [-1.1542e+01, -1.4204e+01, -1.0695e+01,  ..., -5.9768e+00,
           -1.1744e+01, -4.3395e+00],
          [-9.0177e+00, -1.3994e+01, -1.1905e+01,  ..., -4.0330e+00,
           -7.0335e+00, -7.5013e-01],
          [-7.9449e+00, -5.0412e+00, -4.3395e+00,  ..., -6.8561e-01,
           -6.6947e-01, -2.2827e+00]],
conv_ref = torch.nn.Conv2d(in_channels = 16, out_channels=16, kernel_size = 3, padding=1)
conv_ref.weight = model.features[3].weight_q
conv_ref.bias = model.features[3].bias
output_ref = conv_ref(act)
print(output_ref)
```

```
tensor([[[[-1.5978e+00, -6.5127e+00, -5.5128e+00,  ..., -5.1591e+00,
           -8.7421e+00,  1.3061e-01],
          [-7.9954e-01, -7.7770e+00, -4.2379e+00,  ..., -4.6463e+00,
           -4.2488e+00,  9.7652e-01],
          [-5.0718e+00, -4.9145e+00, -6.3376e+00,  ..., -4.8920e+00,
           -4.0186e+00, -1.4195e+00],
          ...,
          [-1.2219e+01, -1.1925e+01, -1.0580e+01,  ..., -1.8536e+01,
            7.7679e+00, -1.4140e+01],
          [-1.6131e+01, -1.0034e+01, -1.3484e+01,  ..., -1.8274e+00,
           -4.9837e+00, -7.2900e+00],
          [-7.3832e+00, -9.5918e+00, -3.6329e+00,  ...,  7.6079e+00,
           -1.7443e-01,  7.1397e+00]],

         [[ 7.8068e+00,  1.6535e+01,  1.0463e+01,  ...,  1.1326e+01,
            9.2186e+00,  3.4775e+00],
          [ 1.5739e+01,  1.2125e+01,  7.5423e+00,  ...,  5.5177e+00,
            6.8292e-01, -4.1647e+00],
          [ 9.5548e+00,  7.2418e+00,  6.0641e+00,  ...,  6.8435e+00,
            1.1637e+00, -6.5314e-01],
          ...,
          [ 1.1280e+01,  5.7424e+00,  5.8696e+00,  ...,  9.1491e+00,
            2.4242e+00,  1.1633e+00],
          [ 5.6103e+00,  6.4178e+00,  1.0718e+01,  ...,  1.0430e+01,
            4.4242e+00, -9.5425e-01],
          [ 3.4204e+00,  5.2349e+00,  4.2090e+00,  ..., -2.3465e+00,
           -5.1125e+00, -3.0515e+00]],

         [[ 1.2091e+01,  2.0422e+01,  1.9287e+01,  ...,  2.0147e+01,
            2.0377e+01,  1.7282e+01],
          [ 1.2631e+01,  1.8010e+01,  1.5468e+01,  ...,  1.3493e+01,
            1.6666e+01,  1.5477e+01],
          [ 1.0508e+01,  1.3140e+01,  8.7123e+00,  ...,  9.8301e+00,
            1.3664e+01,  1.3855e+01],
          ...,
          [ 2.0046e+01,  2.2709e+01,  2.6273e+01,  ...,  2.5011e+01,
            3.0057e+01,  2.2089e+01],
          [ 2.1343e+01,  2.8581e+01,  3.2931e+01,  ...,  3.2863e+01,
            3.6738e+01,  2.8215e+01],
          [ 2.0628e+01,  2.7869e+01,  3.0592e+01,  ...,  3.0897e+01,
            3.4705e+01,  2.3409e+01]],

         ...,

         [[-1.4721e+01, -1.4709e+01, -1.3003e+01,  ..., -7.5585e+00,
            1.5354e+00,  4.6465e+00],
          [-1.2754e+01, -9.4926e+00, -1.3309e+01,  ..., -6.9194e+00,
            4.4701e+00,  7.6075e-01],
          [-1.0664e+01, -9.9545e+00, -1.2136e+01,  ..., -5.6233e+00,
            4.6432e+00, -1.0462e+00],
          ...,
          [-1.3518e+01, -1.5582e+01, -1.0749e+01,  ..., -7.1461e+00,
           -1.1693e+01, -1.9643e+00],
          [-1.3367e+01, -1.7461e+01, -1.3805e+01,  ..., -4.4659e+00,
           -5.4950e+00,  1.2971e+00],
          [-1.0785e+01, -5.0260e+00, -3.3810e+00,  ...,  1.8637e+00,
            2.6483e-01, -1.8053e+00]],
```

```
# 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
# He hard coded these values :(
#array_size = 64 # row and column number
array_size = w_int.size(0)

nig = range(a_int.size(1))  ## ni group [0,1,...31]
njg = range(a_int.size(2))  ## nj group
```

```
icg = range(int(w_int.size(1)))  ## input channel [0,...63]
ocg = range(int(w_int.size(0)))  ## output channel


kijg = range(w_int.size(2)) # [0, .. 8]
ki_dim = int(math.sqrt(w_int.size(2)))  ## Kernel's 1 dim size

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

```
print(act_int.shape)
print(weight_int.shape)
```

```
torch.Size([512, 64, 32, 32])
torch.Size([64, 64, 3, 3])
```

```
#############################################

p_nijg = range(a_pad.size(1)) ## paded activation's nij group [0, ...34*34-1]

psum = torch.zeros( array_size, len(p_nijg), len(kijg)).cuda()

for kij in kijg:
    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[:,:,kij])
        psum[:, nij, kij] = m(a_pad[:,nij]).cuda()
```

```
import math

a_pad_ni_dim = int(math.sqrt(a_pad.size(1))) # 32 + 2*pad = 34

o_ni_dim = int((a_pad_ni_dim - (ki_dim- 1) - 1)/stride + 1) #34 - 2 - 1 + 1 = 32
o_nijg = range(o_ni_dim**2) # [0, 32*32-1]

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


### SFP accumulation ###
for o_nij in o_nijg:
    for kij in kijg:  #[0, ... 8]
        out[:,o_nij] = out[:,o_nij] + \
        psum[:, 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]
                ## 2nd index = (int(o_nij/30)*32 + o_nij%30) + (int(kij/3)*32 + kij%3)
```

```
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())
```

```
tensor(14.8769, device='cuda:0', grad_fn=<SumBackward0>)
```

```
output_int[0,:,:,:]
```

```
tensor([[[-8.9000e+01, -5.1600e+02, -3.6500e+02,  ..., -3.4700e+02,
          -6.0600e+02,  1.5000e+01],
         [-1.6100e+02, -5.5600e+02, -3.2300e+02,  ..., -3.2600e+02,
          -3.1500e+02,  4.3000e+01],
         [-3.9300e+02, -2.5600e+02, -4.9300e+02,  ..., -3.4900e+02,
          -3.2800e+02, -1.0200e+02],
         ...,
         [-9.0300e+02, -8.0800e+02, -7.5500e+02,  ..., -1.3290e+03,
           4.5600e+02, -1.0630e+03],
         [-1.1820e+03, -6.6500e+02, -9.6600e+02,  ..., -1.2900e+02,
          -3.9900e+02, -6.2800e+02],
         [-5.7500e+02, -5.8900e+02, -2.5200e+02,  ...,  5.2300e+02,
          -7.2000e+01,  4.5900e+02]],

        [[ 5.9800e+02,  1.1620e+03,  7.8200e+02,  ...,  8.0200e+02,
           6.6300e+02,  2.8200e+02],
         [ 1.1190e+03,  7.7300e+02,  5.2300e+02,  ...,  3.7900e+02,
           7.5000e+01, -2.6900e+02],
         [ 6.7500e+02,  4.6600e+02,  4.2600e+02,  ...,  4.9700e+02,
           1.0900e+02, -2.8000e+01],
         ...,
         [ 7.5300e+02,  4.7400e+02,  4.2500e+02,  ...,  6.5000e+02,
           2.6400e+02,  1.3300e+02],
         [ 4.0800e+02,  5.0300e+02,  8.3100e+02,  ...,  7.3100e+02,
           3.5500e+02, -7.3000e+01],
         [ 2.1100e+02,  3.7800e+02,  3.0600e+02,  ..., -1.8300e+02,
          -4.3300e+02, -1.9400e+02]],

        [[ 8.4000e+02,  1.4140e+03,  1.3110e+03,  ...,  1.4670e+03,
           1.4920e+03,  1.2780e+03],
         [ 8.5100e+02,  1.2530e+03,  1.0710e+03,  ...,  1.0190e+03,
           1.2500e+03,  1.1570e+03],
         [ 7.0400e+02,  9.3500e+02,  6.4400e+02,  ...,  7.8900e+02,
           1.0560e+03,  1.0550e+03],
         ...,
         [ 1.3950e+03,  1.6110e+03,  1.9450e+03,  ...,  1.8940e+03,
           2.1310e+03,  1.5620e+03],
         [ 1.4340e+03,  1.9690e+03,  2.3540e+03,  ...,  2.3370e+03,
           2.5990e+03,  2.0250e+03],
         [ 1.4370e+03,  1.9370e+03,  2.1440e+03,  ...,  2.1420e+03,
           2.4570e+03,  1.6820e+03]],

        ...,
```