[HW6_prob1]_VGGNet_Hardware_Mapping

November 11, 2022

[1]:
```
import argparse
import os
import time
import shutil

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

import torchvision
import torchvision.transforms as transforms

from models import *

global best_prec
use_gpu = torch.cuda.is_available()
print('=> Building model...')


batch_size = 128
model_name = "VGG16_quant"
model = VGG16_quant()
print(model)

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


train_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
```

```
        transforms.ToTensor(),
        normalize,
    ]))
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=2)


test_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        normalize,
    ]))

testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=2)


print_freq = 100 # every 100 batches, accuracy printed. Here, each batch␣
 ↪includes "batch_size" data points
# CIFAR10 has 50,000 training data, and 10,000 validation data.

def train(trainloader, model, criterion, optimizer, epoch):
    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    model.train()

    end = time.time()
    for i, (input, target) in enumerate(trainloader):
        # measure data loading time
        data_time.update(time.time() - end)

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

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

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

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

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


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



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

    # switch to evaluate mode
    model.eval()

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

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

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

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

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

```
            end = time.time()

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

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


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

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

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


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

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

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

```python
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
```

[ ]:

```
# This cell won't be given, but students will complete the training

lr = 4.4e-2
weight_decay = 1e-4
epochs = 60
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
    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)
```

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

model.cuda()
model.eval()

test_loss = 0
correct = 0

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

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

test_loss /= len(testloader.dataset)

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

```
/opt/conda/lib/python3.9/site-packages/torch/nn/functional.py:718: UserWarning:
Named tensors and all their associated APIs are an experimental feature and
subject to change. Please do not use them for anything important until they are
released as stable. (Triggered internally at
/pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation,
ceil_mode)


Test set: Accuracy: 8954/10000 (90%)
```

[3]:
```
class SaveOutput:
    def __init__(self):
        self.outputs = []
    def __call__(self, module, module_in):
        self.outputs.append(module_in)
    def clear(self):
        self.outputs = []

######### Save inputs from selected layer ##########
save_output = SaveOutput()
i = 0

for layer in model.modules():
    i = i+1
    if isinstance(layer, QuantConv2d):
        print(i,"-th layer prehooked")
        layer.register_forward_pre_hook(save_output)
####################################################

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

```
3 -th layer prehooked
7 -th layer prehooked
```

```
12 -th layer prehooked
16 -th layer prehooked
21 -th layer prehooked
25 -th layer prehooked
29 -th layer prehooked
34 -th layer prehooked
38 -th layer prehooked
42 -th layer prehooked
47 -th layer prehooked
51 -th layer prehooked
55 -th layer prehooked
```

[4]:
```
weight_q = model.features[3].weight_q
w_alpha = model.features[3].weight_quant.wgt_alpha
w_bit = 4

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

[5]:
```
act = save_output.outputs[1][0]
act_alpha  = model.features[3].act_alpha
act_bit = 4
act_quant_fn = act_quantization(act_bit)

act_q = act_quant_fn(act, act_alpha)

act_int = act_q / (act_alpha / (2**act_bit-1))
#print(act_int)
```

[6]:
```
conv_int = torch.nn.Conv2d(in_channels = 64, out_channels=64, kernel_size = 3,␣
 ↪padding=1)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
conv_int.bias = model.features[3].bias
output_int = conv_int(act_int)
output_recovered = output_int * (act_alpha / (2**act_bit-1)) * (w_alpha /␣
 ↪(2**(w_bit-1)-1))
#print(output_recovered)
```

[7]:
```
conv_ref = torch.nn.Conv2d(in_channels = 64, out_channels=64, kernel_size = 3,␣
 ↪padding=1)
conv_ref.weight = model.features[3].weight_q
conv_ref.bias = model.features[3].bias
output_ref = conv_ref(act)
#print(output_ref)
```

[8]:
```
# 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])
print(w_int.size())
padding = 1
stride = 1
array_size = 16 # row and column number

nig = range(a_int.size(1))  ## ni group
njg = range(a_int.size(2))  ## nj group

icg = range(int(w_int.size(1)))  ## input channel
ocg = range(int(w_int.size(0)))  ## output channel

ic_tile = range(4)
oc_tile = range(4)


kijg = range(w_int.size(2))
ki_dim = int(math.sqrt(w_int.size(2)))  ## Kernel's 1 dim size

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

```
torch.Size([64, 64, 9])
```

[9]:

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

a_tile = torch.zeros(len(ic_tile),array_size, a_pad.size(1)).cuda()


for ic_tile_q in ic_tile:
    a_tile[ic_tile_q,:,:] = a_pad[array_size*ic_tile_q:
↪array_size*(ic_tile_q+1),:]



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

```
psum = torch.zeros(len(ic_tile), len(oc_tile), array_size, len(p_nijg),␣
 ↪len(kijg)).cuda()

for kij in kijg:
    for ic_tile_q in ic_tile:
        for oc_tile_q in oc_tile:
            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[array_size*oc_tile_q:
 ↪array_size*(oc_tile_q+1),array_size*ic_tile_q:array_size*(ic_tile_q+1),kij])
                psum[ic_tile_q, oc_tile_q,:, nij, kij] = m(a_tile[ic_tile_q, :,␣
 ↪nij]).cuda()
```

[10]:
```
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)


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


### SFP accumulation ###
for o_nij in o_nijg:
    for kij in kijg:
        for ic_tile_q in ic_tile:
            for oc_tile_q in oc_tile:
                out[oc_tile_q*array_size:(oc_tile_q+1)*array_size,o_nij] =␣
 ↪out[oc_tile_q*array_size:(oc_tile_q+1)*array_size,o_nij] + \
                psum[ic_tile_q, oc_tile_q, :, 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]
```

[11]:
```
out_2D = torch.reshape(out, (out.size(0), 32, -1))
difference = (out_2D - output_int[0,:,:,:])
print(difference.sum())
```

```
tensor(-0.0575, device='cuda:0', grad_fn=<SumBackward0>)
```

[12]:
```
output_int[0,:,:,:]
```

[12]:
```
tensor([[[-2.2300e+02, -2.9500e+02, -2.8500e+02,  …, -3.6100e+02,
          -2.4900e+02, -1.3500e+02],
         [-1.8100e+02, -3.2400e+02, -2.4400e+02,  …, -2.7400e+02,
          -1.4400e+02,  9.2000e+01],
```

```
 [-1.6700e+02, -3.1300e+02, -2.6500e+02,  …, -3.5500e+02,
  -2.4600e+02,  1.9000e+01],
 …,
 [-2.9100e+02, -2.7800e+02, -2.0000e+02,  …, -4.2300e+02,
   7.7000e+01, -9.5000e+01],
 [-3.4300e+02, -4.1200e+02, -1.5900e+02,  …, -3.4400e+02,
  -1.5000e+01,  6.0000e+01],
 [-1.7300e+02, -2.5400e+02, -3.3000e+01,  …, -1.2100e+02,
  -1.1900e+02,  1.3000e+02]],

[[-3.3900e+02,  1.6800e+02, -8.4000e+01,  …, -7.7000e+01,
   5.0000e+01,  9.1000e+01],
 [-8.0100e+02, -2.8400e+02, -5.5200e+02,  …, -6.0600e+02,
  -4.4200e+02, -2.8500e+02],
 [-8.3800e+02, -2.3900e+02, -3.9500e+02,  …, -5.8900e+02,
  -3.9700e+02, -2.6600e+02],
 …,
 [-1.0750e+03, -1.8220e+03, -1.3630e+03,  …, -1.6260e+03,
  -1.8430e+03, -6.1000e+02],
 [-1.1260e+03, -1.7480e+03, -1.3330e+03,  …, -1.6850e+03,
  -1.7530e+03, -5.7800e+02],
 [-7.3900e+02, -1.0950e+03, -7.8800e+02,  …, -1.0700e+03,
  -1.0210e+03, -2.9700e+02]],

[[-1.9200e+02,  2.0000e+01,  1.6000e+01,  …,  2.0000e+00,
   7.8000e+01,  2.6400e+02],
 [-9.9000e+01, -9.2000e+01, -2.9600e+02,  …, -6.1000e+01,
   1.2100e+02, -8.5000e+01],
 [-7.8000e+01, -1.2300e+02, -7.1000e+01,  …, -3.8000e+01,
   1.0000e+00, -1.1300e+02],
 …,
 [-3.3900e+02, -4.2000e+01,  2.6000e+02,  …,  4.4100e+02,
  -3.4300e+02, -1.6700e+02],
 [ 2.1700e+02, -6.7000e+01, -2.7200e+02,  …,  4.9600e+02,
  -6.1900e+02,  3.6700e+02],
 [ 3.0000e+00, -7.7000e+01,  1.2000e+01,  …, -1.1800e+02,
   3.7600e+02, -5.8000e+01]],


…,


[[-3.9600e+02, -6.0200e+02, -4.8100e+02,  …, -6.8300e+02,
  -3.9800e+02, -2.0100e+02],
 [-5.7500e+02, -6.5900e+02, -5.4400e+02,  …, -7.9500e+02,
  -2.4700e+02, -6.1000e+01],
 [-6.0100e+02, -6.9800e+02, -5.9500e+02,  …, -7.5700e+02,
  -2.7200e+02, -1.6500e+02],
 …,
```

```
         [-9.6000e+02, -8.0500e+02, -7.3500e+02,  …, -1.0420e+03,
          -4.0000e+01, -5.1000e+01],
         [-1.0040e+03, -9.1700e+02, -5.4200e+02,  …, -1.1160e+03,
          -4.2100e+02, -1.6100e+02],
         [-6.1700e+02, -7.3000e+02, -5.4600e+02,  …, -7.4400e+02,
          -2.4500e+02,  1.0000e+01]],

        [[ 3.4000e+02,  5.1000e+02,  4.5500e+02,  …,  2.7000e+02,
           6.5000e+01, -1.7600e+02],
         [ 4.2000e+02,  1.3400e+02,  9.3000e+01,  …, -1.3400e+02,
          -4.3900e+02, -6.7000e+02],
         [ 2.7200e+02,  5.7000e+01, -2.8000e+01,  …, -3.3500e+02,
          -4.1700e+02, -5.7500e+02],
         …,
         [-6.3400e+02, -7.7300e+02, -6.0400e+02,  …, -9.9900e+02,
          -3.9500e+02,  9.5000e+01],
         [-9.0700e+02, -4.6000e+02, -2.1300e+02,  …, -7.2900e+02,
          -2.8000e+01,  6.2000e+01],
         [-2.5000e+02,  2.8200e+02,  2.3200e+02,  …,  2.0200e+02,
           4.3000e+02,  4.7600e+02]],

        [[-1.7400e+02, -1.3200e+02,  3.8000e+01,  …, -1.3500e+02,
          -1.7000e+02, -1.1200e+02],
         [-3.1500e+02, -4.3600e+02, -3.1500e+02,  …, -1.1200e+02,
          -1.1900e+02, -1.6000e+01],
         [-3.6100e+02, -5.1000e+02, -3.7400e+02,  …, -3.4700e+02,
          -4.0500e+02, -2.4300e+02],
         …,
         [-4.1600e+02, -4.3800e+02, -3.4300e+02,  …, -1.3200e+02,
          -3.8500e+02, -5.7500e+02],
         [-4.7500e+02, -6.9200e+02, -5.7000e+02,  …, -6.2900e+02,
          -8.1900e+02, -5.2300e+02],
         [-1.0700e+02, -1.3900e+02, -2.9000e+01,  …, -1.9600e+02,
          -3.0600e+02, -1.3700e+02]]], device='cuda:0',
       grad_fn=<SliceBackward>)
```

[ ]:
```
######## Easier 2D version ########

import math

kig = range(int(math.sqrt(len(kijg))))
kjg = range(int(math.sqrt(len(kijg))))

o_nig = range(int((math.sqrt(len(nijg))+2*padding -(math.sqrt(len(kijg))- 1) -␣
 ↪1)/stride + 1))
o_njg = range(int((math.sqrt(len(nijg))+2*padding -(math.sqrt(len(kijg)) - 1) -␣
 ↪1)/stride + 1))
```

```
out = torch.zeros(len(ocg), len(o_nig), len(o_njg)).cuda()


### SFP accumulation ###
for ni in o_nig:
    for nj in o_njg:
        for ki in kig:
            for kj in kjg:
                for ic_tile in ic_tileg:
                    for oc_tile in oc_tileg:
                        out[oc_tile*array_size:(oc_tile+1)*array_size, ni, nj]␣
 ↪= out[oc_tile*array_size:(oc_tile+1)*array_size, ni, nj] + \
                        psum[ic_tile, oc_tile, :, int(math.
 ↪sqrt(len(nijg)))*(ni+ki) + (nj+kj), len(kig)*ki+kj]
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```