

[W2S2_example2][HW3_prob1]__CNN_Training_for_CIFAR10

October 17, 2021

```
[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 *    # bring everything in the folder models

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

batch_size = 128

model_name = "vgg"
model = VGG16()

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() ## at the begining of each epoch, this should
↳be reset
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    model.train()

    end = time.time() # measure current time

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

        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) # time spent to process one batch
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    ## n is impact factor
    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...

Files already downloaded and verified

Files already downloaded and verified

```

[2]: import matplotlib.pyplot as plt
import numpy as np

# functions to show an image

def imshow(img):
    img = img / 2 + 0.5    # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

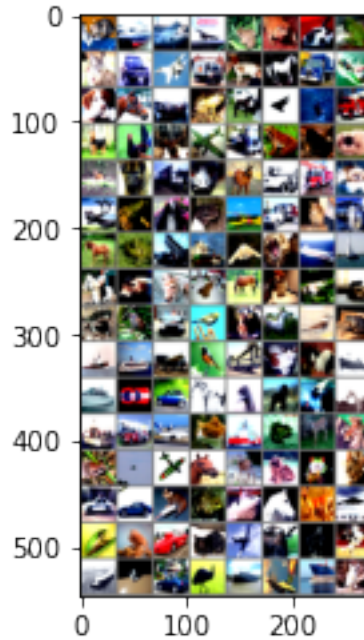
# get some random training images
dataiter = iter(testloader)

```

```
images, labels = dataiter.next() ## If you run this line, the next data batch  
→ is called subsequently.
```

```
# show images  
imshow(torchvision.utils.make_grid(images))
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



```
[ ]: # This cell is from the website  
  
lr = 4e-2  
weight_decay = 1e-4  
epochs = 300  
best_prec = 0  
  
model = model.cuda()  
criterion = nn.CrossEntropyLoss().cuda()  
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9,  
→ weight_decay=weight_decay)  
# weight decay: for regularization to prevent overfitting  
  
if not os.path.exists('result'):  
    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)

```

```

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

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

criterion = nn.CrossEntropyLoss().cuda()

model.eval()
model.cuda()

prec = validate(testloader, model, criterion)

```

/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: [0/79] Time 0.777 (0.777) Loss 0.3099 (0.3099) Prec 91.406%
(91.406%)

* Prec 90.340%

```
[ ]: # HW

# 1. train resnet20 and vgg16 to achieve >90% accuracy
# 2. save your trained model in the result folder
# 3. Restart your jupyter notebook by "Kernel - Restart & Clear Output"
# 4. Load your saved model for vgg16 and validate to see the accuracy
# 5. such as the last part of "[W2S2_example2]_CNN_for_MNIST.ipynb", prehook_
    ↳ the input layers of all the conv layers.
# 6. from the first prehooked input, compute to get the second prehooked input.
    ↳
# 7. Compare your computed second input vs. the prehooked second input.
```

```
[ ]: 
[ ]: 
[ ]: 
[ ]:
```