dl-miniproject-1

April 13, 2024

[ ]:
```python
import torch.nn as nn
import torch.nn.init as init

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

import torchvision
import torchvision.transforms as transforms

from torchsummary import summary
```

[ ]:
```python
'''ResNet in PyTorch.

For Pre-activation ResNet, see 'preact_resnet.py'.

Reference:
[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
    Deep Residual Learning for Image Recognition. arXiv:1512.03385
'''
def _weights_init(m):
    classname = m.__class__.__name__
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        init.kaiming_normal_(m.weight)

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,␣
 ↪padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,stride=1,␣
 ↪padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
```

```
        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out


class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 16

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.layer1 = self._make_layer(block, 16, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 32, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 64, num_blocks[2], stride=2)
        self.linear = nn.Linear(64*block.expansion, num_classes)

        self.apply(_weights_init)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = F.avg_pool2d(out, out.size()[3])
        out = out.view(out.size(0), -1)
        out = self.linear(out)
```

```
        return out


def ResNet68():
    return ResNet(BasicBlock, [11, 11, 11])
```

[ ]:
```
if torch.cuda.is_available():
    device = torch.device('cuda')
    print("CUDA Start!")
else:
    device = torch.device('cpu')
    print("Using CPU")

net = ResNet68()
net = net.to(device)
```

```
CUDA Start!
```

[ ]:
```
# Define transformations for data preprocessing
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])
```

[ ]:
```
# Define batch sizes
train_batch_size = 64  # Reduced batch size
test_batch_size = 64  # Reduced batch size
```

[ ]:
```
# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=train_batch_size, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=test_batch_size, shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat', 'deer',
```

```
            'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
best_acc = 0  # best test accuracy
start_epoch = 0  # start from epoch 0
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01,
                      momentum=0.9, weight_decay=5e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=50)

# Lists to store final results of each epoch
train_accuracies = []
test_accuracies= []
train_losses = []
test_losses = []

# Training function
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0

    # #Defining learning rate
    # if epoch < warmup_epochs:
    #     # Adjust learning rate for warm-up epochs
    #     warmup_factor = (epoch + 1) / warmup_epochs
    #     curr_lr = warmup_lr_init + (warmup_lr_final - warmup_lr_init) *␣
↪warmup_factor
    #     for param_group in optimizer.param_groups:
    #         param_group['lr'] = curr_lr
    # else:
    #     # Revert to original learning rate after warm-up
    #     for param_group in optimizer.param_groups:
    #         param_group['lr'] = warmup_lr_final

    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
```

```python
        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

    # Calculate and save final accuracy for this epoch
    train_accuracy = 100. * correct / total
    train_accuracies.append(train_accuracy)

    train_loss /= len(trainloader)  # Average loss per batch
    train_losses.append(train_loss)

    print("train accuracy: ", train_accuracy)
    print("train loss: ", train_loss)

# Testing function
def test(epoch):
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(testloader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    # Calculate and save final accuracy for this epoch
    test_accuracy = 100. * correct / total
    test_accuracies.append(test_accuracy)
    test_loss /= len(testloader)  # Average loss per batch
    test_losses.append(test_loss)

    print("test accuracy: ", test_accuracy)
    print("test loss: ", test_loss)
```

[ ]:
```python
summary(net, input_size=(3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
```

```
================================================================
            Conv2d-1           [-1, 16, 32, 32]             432
       BatchNorm2d-2           [-1, 16, 32, 32]              32
            Conv2d-3           [-1, 16, 32, 32]           2,304
       BatchNorm2d-4           [-1, 16, 32, 32]              32
            Conv2d-5           [-1, 16, 32, 32]           2,304
       BatchNorm2d-6           [-1, 16, 32, 32]              32
        BasicBlock-7           [-1, 16, 32, 32]               0
            Conv2d-8           [-1, 16, 32, 32]           2,304
       BatchNorm2d-9           [-1, 16, 32, 32]              32
           Conv2d-10           [-1, 16, 32, 32]           2,304
      BatchNorm2d-11           [-1, 16, 32, 32]              32
       BasicBlock-12           [-1, 16, 32, 32]               0
           Conv2d-13           [-1, 16, 32, 32]           2,304
      BatchNorm2d-14           [-1, 16, 32, 32]              32
           Conv2d-15           [-1, 16, 32, 32]           2,304
      BatchNorm2d-16           [-1, 16, 32, 32]              32
       BasicBlock-17           [-1, 16, 32, 32]               0
           Conv2d-18           [-1, 16, 32, 32]           2,304
      BatchNorm2d-19           [-1, 16, 32, 32]              32
           Conv2d-20           [-1, 16, 32, 32]           2,304
      BatchNorm2d-21           [-1, 16, 32, 32]              32
       BasicBlock-22           [-1, 16, 32, 32]               0
           Conv2d-23           [-1, 16, 32, 32]           2,304
      BatchNorm2d-24           [-1, 16, 32, 32]              32
           Conv2d-25           [-1, 16, 32, 32]           2,304
      BatchNorm2d-26           [-1, 16, 32, 32]              32
       BasicBlock-27           [-1, 16, 32, 32]               0
           Conv2d-28           [-1, 16, 32, 32]           2,304
      BatchNorm2d-29           [-1, 16, 32, 32]              32
           Conv2d-30           [-1, 16, 32, 32]           2,304
      BatchNorm2d-31           [-1, 16, 32, 32]              32
       BasicBlock-32           [-1, 16, 32, 32]               0
           Conv2d-33           [-1, 16, 32, 32]           2,304
      BatchNorm2d-34           [-1, 16, 32, 32]              32
           Conv2d-35           [-1, 16, 32, 32]           2,304
      BatchNorm2d-36           [-1, 16, 32, 32]              32
       BasicBlock-37           [-1, 16, 32, 32]               0
           Conv2d-38           [-1, 16, 32, 32]           2,304
      BatchNorm2d-39           [-1, 16, 32, 32]              32
           Conv2d-40           [-1, 16, 32, 32]           2,304
      BatchNorm2d-41           [-1, 16, 32, 32]              32
       BasicBlock-42           [-1, 16, 32, 32]               0
           Conv2d-43           [-1, 16, 32, 32]           2,304
      BatchNorm2d-44           [-1, 16, 32, 32]              32
           Conv2d-45           [-1, 16, 32, 32]           2,304
      BatchNorm2d-46           [-1, 16, 32, 32]              32
       BasicBlock-47           [-1, 16, 32, 32]               0
```

Conv2d-48	[-1, 16, 32, 32]	2,304
BatchNorm2d-49	[-1, 16, 32, 32]	32
Conv2d-50	[-1, 16, 32, 32]	2,304
BatchNorm2d-51	[-1, 16, 32, 32]	32
BasicBlock-52	[-1, 16, 32, 32]	0
Conv2d-53	[-1, 16, 32, 32]	2,304
BatchNorm2d-54	[-1, 16, 32, 32]	32
Conv2d-55	[-1, 16, 32, 32]	2,304
BatchNorm2d-56	[-1, 16, 32, 32]	32
BasicBlock-57	[-1, 16, 32, 32]	0
Conv2d-58	[-1, 32, 16, 16]	4,608
BatchNorm2d-59	[-1, 32, 16, 16]	64
Conv2d-60	[-1, 32, 16, 16]	9,216
BatchNorm2d-61	[-1, 32, 16, 16]	64
Conv2d-62	[-1, 32, 16, 16]	512
BatchNorm2d-63	[-1, 32, 16, 16]	64
BasicBlock-64	[-1, 32, 16, 16]	0
Conv2d-65	[-1, 32, 16, 16]	9,216
BatchNorm2d-66	[-1, 32, 16, 16]	64
Conv2d-67	[-1, 32, 16, 16]	9,216
BatchNorm2d-68	[-1, 32, 16, 16]	64
BasicBlock-69	[-1, 32, 16, 16]	0
Conv2d-70	[-1, 32, 16, 16]	9,216
BatchNorm2d-71	[-1, 32, 16, 16]	64
Conv2d-72	[-1, 32, 16, 16]	9,216
BatchNorm2d-73	[-1, 32, 16, 16]	64
BasicBlock-74	[-1, 32, 16, 16]	0
Conv2d-75	[-1, 32, 16, 16]	9,216
BatchNorm2d-76	[-1, 32, 16, 16]	64
Conv2d-77	[-1, 32, 16, 16]	9,216
BatchNorm2d-78	[-1, 32, 16, 16]	64
BasicBlock-79	[-1, 32, 16, 16]	0
Conv2d-80	[-1, 32, 16, 16]	9,216
BatchNorm2d-81	[-1, 32, 16, 16]	64
Conv2d-82	[-1, 32, 16, 16]	9,216
BatchNorm2d-83	[-1, 32, 16, 16]	64
BasicBlock-84	[-1, 32, 16, 16]	0
Conv2d-85	[-1, 32, 16, 16]	9,216
BatchNorm2d-86	[-1, 32, 16, 16]	64
Conv2d-87	[-1, 32, 16, 16]	9,216
BatchNorm2d-88	[-1, 32, 16, 16]	64
BasicBlock-89	[-1, 32, 16, 16]	0
Conv2d-90	[-1, 32, 16, 16]	9,216
BatchNorm2d-91	[-1, 32, 16, 16]	64
Conv2d-92	[-1, 32, 16, 16]	9,216
BatchNorm2d-93	[-1, 32, 16, 16]	64
BasicBlock-94	[-1, 32, 16, 16]	0
Conv2d-95	[-1, 32, 16, 16]	9,216

BatchNorm2d-96	[-1, 32, 16, 16]	64
Conv2d-97	[-1, 32, 16, 16]	9,216
BatchNorm2d-98	[-1, 32, 16, 16]	64
BasicBlock-99	[-1, 32, 16, 16]	0
Conv2d-100	[-1, 32, 16, 16]	9,216
BatchNorm2d-101	[-1, 32, 16, 16]	64
Conv2d-102	[-1, 32, 16, 16]	9,216
BatchNorm2d-103	[-1, 32, 16, 16]	64
BasicBlock-104	[-1, 32, 16, 16]	0
Conv2d-105	[-1, 32, 16, 16]	9,216
BatchNorm2d-106	[-1, 32, 16, 16]	64
Conv2d-107	[-1, 32, 16, 16]	9,216
BatchNorm2d-108	[-1, 32, 16, 16]	64
BasicBlock-109	[-1, 32, 16, 16]	0
Conv2d-110	[-1, 32, 16, 16]	9,216
BatchNorm2d-111	[-1, 32, 16, 16]	64
Conv2d-112	[-1, 32, 16, 16]	9,216
BatchNorm2d-113	[-1, 32, 16, 16]	64
BasicBlock-114	[-1, 32, 16, 16]	0
Conv2d-115	[-1, 64, 8, 8]	18,432
BatchNorm2d-116	[-1, 64, 8, 8]	128
Conv2d-117	[-1, 64, 8, 8]	36,864
BatchNorm2d-118	[-1, 64, 8, 8]	128
Conv2d-119	[-1, 64, 8, 8]	2,048
BatchNorm2d-120	[-1, 64, 8, 8]	128
BasicBlock-121	[-1, 64, 8, 8]	0
Conv2d-122	[-1, 64, 8, 8]	36,864
BatchNorm2d-123	[-1, 64, 8, 8]	128
Conv2d-124	[-1, 64, 8, 8]	36,864
BatchNorm2d-125	[-1, 64, 8, 8]	128
BasicBlock-126	[-1, 64, 8, 8]	0
Conv2d-127	[-1, 64, 8, 8]	36,864
BatchNorm2d-128	[-1, 64, 8, 8]	128
Conv2d-129	[-1, 64, 8, 8]	36,864
BatchNorm2d-130	[-1, 64, 8, 8]	128
BasicBlock-131	[-1, 64, 8, 8]	0
Conv2d-132	[-1, 64, 8, 8]	36,864
BatchNorm2d-133	[-1, 64, 8, 8]	128
Conv2d-134	[-1, 64, 8, 8]	36,864
BatchNorm2d-135	[-1, 64, 8, 8]	128
BasicBlock-136	[-1, 64, 8, 8]	0
Conv2d-137	[-1, 64, 8, 8]	36,864
BatchNorm2d-138	[-1, 64, 8, 8]	128
Conv2d-139	[-1, 64, 8, 8]	36,864
BatchNorm2d-140	[-1, 64, 8, 8]	128
BasicBlock-141	[-1, 64, 8, 8]	0
Conv2d-142	[-1, 64, 8, 8]	36,864
BatchNorm2d-143	[-1, 64, 8, 8]	128

```
        Conv2d-144             [-1, 64, 8, 8]          36,864
   BatchNorm2d-145             [-1, 64, 8, 8]             128
    BasicBlock-146             [-1, 64, 8, 8]               0
        Conv2d-147             [-1, 64, 8, 8]          36,864
   BatchNorm2d-148             [-1, 64, 8, 8]             128
        Conv2d-149             [-1, 64, 8, 8]          36,864
   BatchNorm2d-150             [-1, 64, 8, 8]             128
    BasicBlock-151             [-1, 64, 8, 8]               0
        Conv2d-152             [-1, 64, 8, 8]          36,864
   BatchNorm2d-153             [-1, 64, 8, 8]             128
        Conv2d-154             [-1, 64, 8, 8]          36,864
   BatchNorm2d-155             [-1, 64, 8, 8]             128
    BasicBlock-156             [-1, 64, 8, 8]               0
        Conv2d-157             [-1, 64, 8, 8]          36,864
   BatchNorm2d-158             [-1, 64, 8, 8]             128
        Conv2d-159             [-1, 64, 8, 8]          36,864
   BatchNorm2d-160             [-1, 64, 8, 8]             128
    BasicBlock-161             [-1, 64, 8, 8]               0
        Conv2d-162             [-1, 64, 8, 8]          36,864
   BatchNorm2d-163             [-1, 64, 8, 8]             128
        Conv2d-164             [-1, 64, 8, 8]          36,864
   BatchNorm2d-165             [-1, 64, 8, 8]             128
    BasicBlock-166             [-1, 64, 8, 8]               0
        Conv2d-167             [-1, 64, 8, 8]          36,864
   BatchNorm2d-168             [-1, 64, 8, 8]             128
        Conv2d-169             [-1, 64, 8, 8]          36,864
   BatchNorm2d-170             [-1, 64, 8, 8]             128
    BasicBlock-171             [-1, 64, 8, 8]               0
        Linear-172                   [-1, 10]             650
================================================================
Total params: 1,050,202
Trainable params: 1,050,202
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 12.47
Params size (MB): 4.01
Estimated Total Size (MB): 16.49
----------------------------------------------------------------
```

[ ]:

```
for epoch in range(start_epoch, start_epoch+50):
    train(epoch)
    test(epoch)
    scheduler.step()
```

```
Epoch: 0
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

```
train accuracy:  35.532
train loss:  1.7703219482965786
test accuracy:  51.1
test loss:  1.3821087726362191

Epoch: 1
train accuracy:  53.044
train loss:  1.3023951265513134
test accuracy:  59.92
test loss:  1.128989104252712

Epoch: 2
train accuracy:  61.53
train loss:  1.0808293226429873
test accuracy:  63.09
test loss:  1.0588979436333772

Epoch: 3
train accuracy:  67.16
train loss:  0.9346146753910557
test accuracy:  65.7
test loss:  1.0048029358219948

Epoch: 4
train accuracy:  70.628
train loss:  0.8345660863020231
test accuracy:  69.57
test loss:  0.9003689482713201

Epoch: 5
train accuracy:  73.94
train loss:  0.7452318467523741
test accuracy:  74.42
test loss:  0.7415300907602735

Epoch: 6
train accuracy:  76.378
train loss:  0.683434475032265
test accuracy:  75.5
test loss:  0.7186958751860698

Epoch: 7
train accuracy:  78.236
```

```
train loss:  0.6283918646976466
test accuracy:  77.07
test loss:  0.6803043144903366

Epoch: 8
train accuracy:  79.536
train loss:  0.5858416350753716
test accuracy:  78.11
test loss:  0.6586965903355058

Epoch: 9
train accuracy:  81.052
train loss:  0.5492438032003619
test accuracy:  77.92
test loss:  0.6551864929259963

Epoch: 10
train accuracy:  82.118
train loss:  0.5197225670947139
test accuracy:  80.97
test loss:  0.5614552488372584

Epoch: 11
train accuracy:  82.842
train loss:  0.4943404701893287
test accuracy:  81.88
test loss:  0.5381744951958869

Epoch: 12
train accuracy:  83.906
train loss:  0.4654122637894452
test accuracy:  82.56
test loss:  0.5154427673406662

Epoch: 13
train accuracy:  84.186
train loss:  0.4519013012271098
test accuracy:  81.91
test loss:  0.5373093344413551

Epoch: 14
train accuracy:  85.024
train loss:  0.42902711887493766
test accuracy:  82.26
test loss:  0.539973954201504

Epoch: 15
train accuracy:  85.958
```

```
train loss:  0.4060071468391382
test accuracy:  84.49
test loss:  0.45643974147784483

Epoch: 16
train accuracy:  86.59
train loss:  0.3883736696656403
test accuracy:  83.59
test loss:  0.5021985254849598

Epoch: 17
train accuracy:  87.08
train loss:  0.3723302806925286
test accuracy:  83.9
test loss:  0.4860828176235697

Epoch: 18
train accuracy:  87.59
train loss:  0.35698250014230115
test accuracy:  84.47
test loss:  0.4746239313464256

Epoch: 19
train accuracy:  88.206
train loss:  0.3407753211496126
test accuracy:  83.43
test loss:  0.5005138052307117

Epoch: 20
train accuracy:  88.674
train loss:  0.33060507566841973
test accuracy:  84.98
test loss:  0.45945999880505217

Epoch: 21
train accuracy:  89.23
train loss:  0.31316830144475793
test accuracy:  84.97
test loss:  0.45294702100525996

Epoch: 22
train accuracy:  89.658
train loss:  0.301171916839488
test accuracy:  86.18
test loss:  0.4113404744654704

Epoch: 23
train accuracy:  89.984
```

```
train loss:  0.2891983661867316
test accuracy:  84.89
test loss:  0.45395589453779206

Epoch: 24
train accuracy:  90.66
train loss:  0.2707700556063134
test accuracy:  86.5
test loss:  0.4167344725815354

Epoch: 25
train accuracy:  90.86
train loss:  0.26429116078045056
test accuracy:  86.68
test loss:  0.40676456177310577

Epoch: 26
train accuracy:  91.294
train loss:  0.24950842771326642
test accuracy:  86.79
test loss:  0.4048594063634326

Epoch: 27
train accuracy:  91.928
train loss:  0.2346901160610073
test accuracy:  87.76
test loss:  0.37989824251004845

Epoch: 28
train accuracy:  92.188
train loss:  0.22129938142645694
test accuracy:  87.81
test loss:  0.38357632227574184

Epoch: 29
train accuracy:  92.672
train loss:  0.21011765463673093
test accuracy:  87.33
test loss:  0.40183076390605066

Epoch: 30
train accuracy:  93.322
train loss:  0.1934369243300327
test accuracy:  87.57
test loss:  0.38906378588479035

Epoch: 31
train accuracy:  93.596
```

```
train loss:  0.18360666210389198
test accuracy:  88.47
test loss:  0.3664971804542906

Epoch: 32
train accuracy:  93.994
train loss:  0.17397073985737227
test accuracy:  88.45
test loss:  0.3737812750753324

Epoch: 33
train accuracy:  94.48
train loss:  0.16196171484430275
test accuracy:  88.46
test loss:  0.37432311195286977

Epoch: 34
train accuracy:  94.784
train loss:  0.14958835134039755
test accuracy:  88.77
test loss:  0.3758160496593281

Epoch: 35
train accuracy:  95.306
train loss:  0.13588695587052027
test accuracy:  88.87
test loss:  0.3757723251440723

Epoch: 36
train accuracy:  95.646
train loss:  0.12709615521771295
test accuracy:  89.45
test loss:  0.36213001089206165

Epoch: 37
train accuracy:  96.202
train loss:  0.11160076887863676
test accuracy:  89.45
test loss:  0.3604400647199078

Epoch: 38
train accuracy:  96.504
train loss:  0.10273157899765789
test accuracy:  89.61
test loss:  0.35670878154457

Epoch: 39
train accuracy:  96.798
```

```
train loss:  0.0944776990655762
test accuracy:  89.4
test loss:  0.36872670251377826

Epoch: 40
train accuracy:  97.146
train loss:  0.08374142051314759
test accuracy:  89.71
test loss:  0.3607630651135733

Epoch: 41
train accuracy:  97.504
train loss:  0.07536501770653307
test accuracy:  89.79
test loss:  0.3655861362245432

Epoch: 42
train accuracy:  97.702
train loss:  0.06931659279515029
test accuracy:  89.53
test loss:  0.3717409841906113

Epoch: 43
train accuracy:  97.988
train loss:  0.06294447509155078
test accuracy:  90.01
test loss:  0.3786829335342167

Epoch: 44
train accuracy:  98.078
train loss:  0.06058408956453109
test accuracy:  90.01
test loss:  0.36903686973319694

Epoch: 45
train accuracy:  98.152
train loss:  0.058057463065128004
test accuracy:  90.07
test loss:  0.36574396343937343

Epoch: 46
train accuracy:  98.134
train loss:  0.057276673431572556
test accuracy:  90.03
test loss:  0.37011214635174744

Epoch: 47
train accuracy:  98.402
```

```
train loss:  0.05175255004035504
test accuracy:  90.04
test loss:  0.36679586438331635

Epoch: 48
train accuracy:  98.454
train loss:  0.05083180782254166
test accuracy:  90.29
test loss:  0.3634474926930704

Epoch: 49
train accuracy:  98.508
train loss:  0.04891194459741645
test accuracy:  90.14
test loss:  0.36804010189927305
```

[ ]:

```python
correct = 0
total = 0
with torch.no_grad():
    for images, labels in testloader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print('Final Accuracy on Test Dataset: {:.2f}%'.format(accuracy))
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

```
Final Accuracy on Test Dataset: 90.14%
```

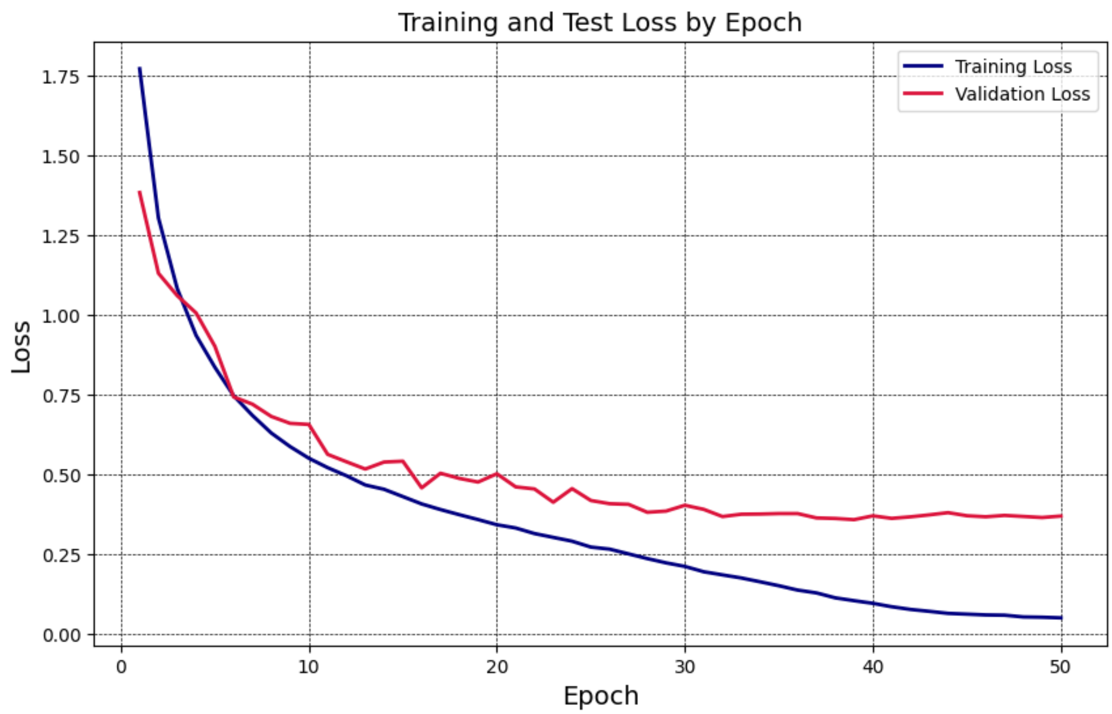
[ ]:

```python
import matplotlib.pyplot as plt
```

[ ]:

```python
fig, ax = plt.subplots(figsize=(10, 6))
epochs = range(1, 51)

# Plotting the training loss
train_line, = ax.plot(epochs, train_losses[:50], label='Training Loss',␣
 ↪color='navy', linewidth=2)

# Plotting the validation loss
```

```
test_line, = ax.plot(epochs, test_losses[:50], label='Validation Loss',␣
 ↪color='crimson', linewidth=2)

ax.set_title('Training and Test Loss by Epoch', fontsize=14)
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('Loss', fontsize=14)
ax.legend(handles=[train_line, test_line], loc='upper right')
ax.grid(True, which='major', linestyle='--', linewidth='0.5', color='black')
ax.grid(True, which='minor', linestyle=':', linewidth='0.5', color='gray')

plt.show()
```



[ ]:
```
fig, ax = plt.subplots(figsize=(10, 6))
epochs = range(1, 51)

# Plotting the training loss
train_line, = ax.plot(epochs, train_accuracies, label='Training Accuracy',␣
 ↪color='navy', linewidth=2)

# Plotting the validation loss
test_line, = ax.plot(epochs, test_accuracies, label='Validation Accuracy',␣
 ↪color='crimson', linewidth=2)

ax.set_title('Training and Test Accuracy by Epoch', fontsize=14)
```

```python
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('Accuracy', fontsize=14)
ax.legend(handles=[train_line, test_line], loc='upper right')
ax.grid(True, which='major', linestyle='--', linewidth='0.5', color='black')
ax.grid(True, which='minor', linestyle=':', linewidth='0.5', color='gray')

plt.show()
```