```
import torch
import torchvision
```

```
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import random_split, DataLoader
```

```
def get_dataloaders(batch_size=32, num_workers=4):
    transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))
            ])

    full_train = datasets.CIFAR10('./data', train=True, download=True, transform=transform)
    test = datasets.CIFAR10('./data', train=False, transform=transform)

    val_size = int(len(full_train) * 0.1)
    train_size = len(full_train) - val_size
    train, val = random_split(full_train, [train_size, val_size])

    train_loader = DataLoader(train, batch_size=batch_size, shuffle=True, num_workers=num_workers)
    val_loader = DataLoader(val, batch_size=batch_size, shuffle=False, num_workers=num_workers)
    test_loader = DataLoader(test, batch_size=batch_size, shuffle=False, num_workers=num_workers)

    classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    return train_loader, val_loader, test_loader
```

```
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import resnet34, ResNet34_Weights
```

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    total_loss = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0) # gradient clipping
        optimizer.step()

        current_sample = len(data)
        total_sample = len(train_loader.dataset)
        total_batch = len(train_loader)
        total_loss += loss.item()

        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * current_sample, total_sample,
                100. * batch_idx / total_batch, loss.item()))
            if args.dry_run:
                break

    return total_loss / total_batch
```

```
def test(model, device, test_loader):
    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)
            output = model(data)
            test_loss += F.cross_entropy(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    total_test_sample = len(test_loader.dataset)

    test_loss /= total_test_sample
    accuracy = 100. * correct / total_test_sample

    print('\nAverage loss: {:.4f}, Accuracy: {:.2f}%\n'.format(test_loss, accuracy))
```

```
    return accuracy
```

```
class Model(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.model = resnet34(weights=ResNet34_Weights.DEFAULT)
        self.model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.model.maxpool = nn.Identity()
        self.model.fc = nn.Linear(self.model.fc.in_features, num_classes)

    def forward(self, x):
        return self.model(x)

model = Model()
```

```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
100%|██████████| 83.3M/83.3M [00:00<00:00, 170MB/s]
```

```
import argparse
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
```

```
parser = argparse.ArgumentParser(description='PyTorch CIFAR10 Practice')
parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                    help='input batch size for training (default: 64)')
parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                    help='input batch size for testing (default: 1000)')
parser.add_argument('--epochs', type=int, default=12, metavar='N',
                    help='number of epochs to train (default: 12)')
parser.add_argument('--lr', type=float, default=0.001, metavar='LR',
                    help='learning rate (default: 0.001)')
parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                    help='Learning rate step gamma (default: 0.7)')
parser.add_argument('--no-accel', action='store_true',
                    help='disables accelerator')
parser.add_argument('--dry-run', action='store_true',
                    help='quickly check a single pass')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
parser.add_argument('--save-model', action='store_true',
                    help='For Saving the current Model')

args = parser.parse_args(args=[])

torch.manual_seed(args.seed)
```

```
<torch._C.Generator at 0x7e4954981c30>
```

```
use_cuda = not args.no_accel and torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
```

```
train_loader, val_loader, test_loader = get_dataloaders(batch_size=args.batch_size, num_workers=2)
```

```
100%|██████████| 170M/170M [00:04<00:00, 41.4MB/s]
```

```
model = Model().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=0.9, weight_decay=5e-4)
scheduler = StepLR(optimizer, step_size=10, gamma=args.gamma)
```

```
train_losses = []
val_accuracies = []

print("=== Train accuracy ===")
for epoch in range(1, args.epochs + 1):
        loss = train(args, model, device, train_loader, optimizer, epoch)
        val_accuracy = test(model, device, val_loader)
        train_losses.append(loss)
        val_accuracies.append(val_accuracy)
        scheduler.step()

print("===Test accuracy===")
test_accuracy = test(model, device, test_loader)
```

```
Train Epoch: 12 [12800/45000 (28%)]     Loss: 0.068449
Train Epoch: 12 [13440/45000 (30%)]     Loss: 0.106245
Train Epoch: 12 [14080/45000 (31%)]     Loss: 0.065969
Train Epoch: 12 [14720/45000 (33%)]     Loss: 0.210170
Train Epoch: 12 [15360/45000 (34%)]     Loss: 0.085418
Train Epoch: 12 [16000/45000 (36%)]     Loss: 0.094945
Train Epoch: 12 [16640/45000 (37%)]     Loss: 0.092067
Train Epoch: 12 [17280/45000 (38%)]     Loss: 0.074912
Train Epoch: 12 [17920/45000 (40%)]     Loss: 0.130951
Train Epoch: 12 [18560/45000 (41%)]     Loss: 0.121490
Train Epoch: 12 [19200/45000 (43%)]     Loss: 0.182971
Train Epoch: 12 [19840/45000 (44%)]     Loss: 0.099104
Train Epoch: 12 [20480/45000 (45%)]     Loss: 0.071300
Train Epoch: 12 [21120/45000 (47%)]     Loss: 0.160613
Train Epoch: 12 [21760/45000 (48%)]     Loss: 0.065559
Train Epoch: 12 [22400/45000 (50%)]     Loss: 0.074144
Train Epoch: 12 [23040/45000 (51%)]     Loss: 0.233769
Train Epoch: 12 [23680/45000 (53%)]     Loss: 0.116715
Train Epoch: 12 [24320/45000 (54%)]     Loss: 0.072764
Train Epoch: 12 [24960/45000 (55%)]     Loss: 0.062236
Train Epoch: 12 [25600/45000 (57%)]     Loss: 0.150772
Train Epoch: 12 [26240/45000 (58%)]     Loss: 0.070140
Train Epoch: 12 [26880/45000 (60%)]     Loss: 0.244628
Train Epoch: 12 [27520/45000 (61%)]     Loss: 0.164064
Train Epoch: 12 [28160/45000 (62%)]     Loss: 0.141191
Train Epoch: 12 [28800/45000 (64%)]     Loss: 0.103403
Train Epoch: 12 [29440/45000 (65%)]     Loss: 0.090053
Train Epoch: 12 [30080/45000 (67%)]     Loss: 0.164904
Train Epoch: 12 [30720/45000 (68%)]     Loss: 0.160456
Train Epoch: 12 [31360/45000 (70%)]     Loss: 0.255905
Train Epoch: 12 [32000/45000 (71%)]     Loss: 0.165514
Train Epoch: 12 [32640/45000 (72%)]     Loss: 0.093133
Train Epoch: 12 [33280/45000 (74%)]     Loss: 0.119281
Train Epoch: 12 [33920/45000 (75%)]     Loss: 0.058372
Train Epoch: 12 [34560/45000 (77%)]     Loss: 0.141375
Train Epoch: 12 [35200/45000 (78%)]     Loss: 0.144516
Train Epoch: 12 [35840/45000 (80%)]     Loss: 0.123774
Train Epoch: 12 [36480/45000 (81%)]     Loss: 0.077357
Train Epoch: 12 [37120/45000 (82%)]     Loss: 0.082682
Train Epoch: 12 [37760/45000 (84%)]     Loss: 0.098814
Train Epoch: 12 [38400/45000 (85%)]     Loss: 0.074840
Train Epoch: 12 [39040/45000 (87%)]     Loss: 0.122709
Train Epoch: 12 [39680/45000 (88%)]     Loss: 0.100422
Train Epoch: 12 [40320/45000 (89%)]     Loss: 0.066433
Train Epoch: 12 [40960/45000 (91%)]     Loss: 0.157345
Train Epoch: 12 [41600/45000 (92%)]     Loss: 0.144861
Train Epoch: 12 [42240/45000 (94%)]     Loss: 0.138848
Train Epoch: 12 [42880/45000 (95%)]     Loss: 0.092020
Train Epoch: 12 [43520/45000 (97%)]     Loss: 0.108647
Train Epoch: 12 [44160/45000 (98%)]     Loss: 0.117876
Train Epoch: 12 [44800/45000 (99%)]     Loss: 0.157560

Average loss: 0.3057, Accuracy: 89.78%

===Test accuracy===

Average loss: 0.3240, Accuracy: 89.88%
```

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

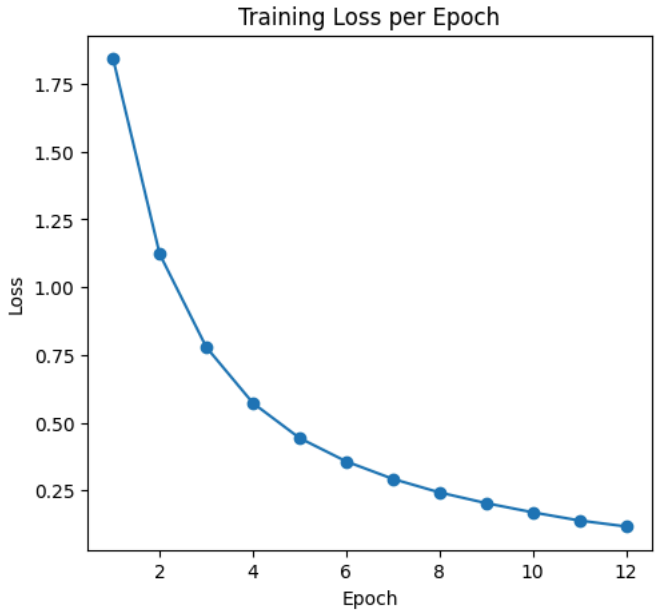
```
epochs = list(range(1, args.epochs+1))
test_accuracy_list = [test_accuracy] * len(epochs)
```

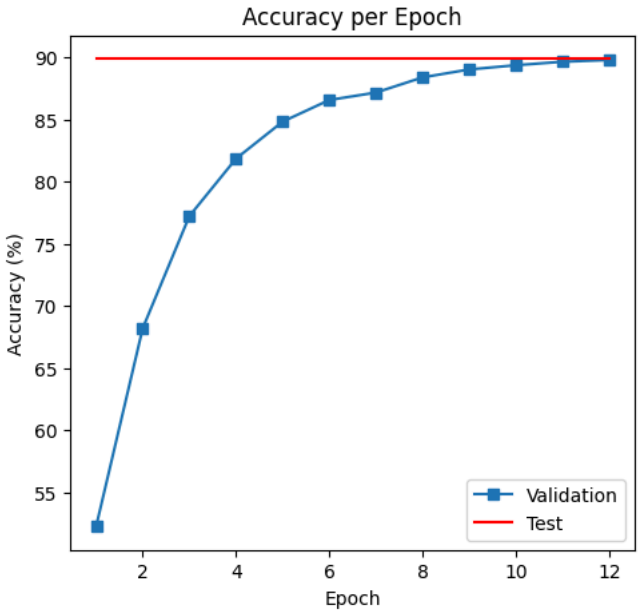
```
plt.figure(figsize=(12, 5))

# Loss
plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, marker='o')
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")

# Accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, val_accuracies, marker='s')
plt.plot(epochs, test_accuracy_list, 'r-')
plt.title("Accuracy per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.legend(['Validation', 'Test'])

plt.show()
```





코딩을 시작하거나 AI로 코드를 생성하세요.