a3_part1_rotation

April 3, 2024

1 (Optional) Colab Setup

If you aren't using Colab, you can delete the following code cell. This is just to help students with mounting to Google Drive to access the other .py files and downloading the data, which is a little trickier on Colab than on your local machine using Jupyter.

[1]:
```
# you will be prompted with a window asking to grant permissions
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

[2]:
```
# fill in the path in your Google Drive in the string below. Note: do not␣
 ↪escape slashes or spaces
import os
datadir = "/content/drive/MyDrive/assignment3_part1/"
if not os.path.exists(datadir):
  !ln -s "/content/drive/MyDrive/assignment3_part1/" $datadir # TODO: Fill your␣
 ↪Assignment 3 path
os.chdir(datadir)
!pwd
```

```
/content/drive/MyDrive/assignment3_part1
```

#Data Setup

The first thing to do is implement a dataset class to load rotated CIFAR10 images with matching labels. Since there is already a CIFAR10 dataset class implemented in `torchvision`, we will extend this class and modify the `__getitem__` method appropriately to load rotated images.

Each rotation label should be an integer in the set {0, 1, 2, 3} which correspond to rotations of 0, 90, 180, or 270 degrees respectively.

[1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np
import random

def rotate_img(img, rot):
```

```
    if rot == 0: # 0 degrees rotation
        return img
    # TODO: Implement rotate_img() - return the rotated img
    elif rot == 1:
        img_rot = transforms.functional.rotate(img, 90)
        return img_rot
    elif rot == 2:
        img_rot = transforms.functional.rotate(img, 180)
        return img_rot
    elif rot == 3:
        img_rot = transforms.functional.rotate(img, 270)
        return img_rot
    else:
        raise ValueError('rotation should be 0, 90, 180, or 270 degrees')

class CIFAR10Rotation(torchvision.datasets.CIFAR10):

    def __init__(self, root, train, download, transform) -> None:
        super().__init__(root=root, train=train, download=download,␣
 ↪transform=transform)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index: int):
        image, cls_label = super().__getitem__(index)

        # randomly select image rotation
        rotation_label = random.choice([0, 1, 2, 3])
        image_rotated = rotate_img(image, rotation_label)

        rotation_label = torch.tensor(rotation_label).long()
        return image, image_rotated, rotation_label, torch.tensor(cls_label).
 ↪long()
```

[2]:
```
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)),
])
```

```
batch_size = 128

trainset = CIFAR10Rotation(root='./data', train=True,
                                          download=True,␣
 ↪transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                           shuffle=True, num_workers=2)

testset = CIFAR10Rotation(root='./data', train=False,
                                         download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                           shuffle=False, num_workers=2)
```

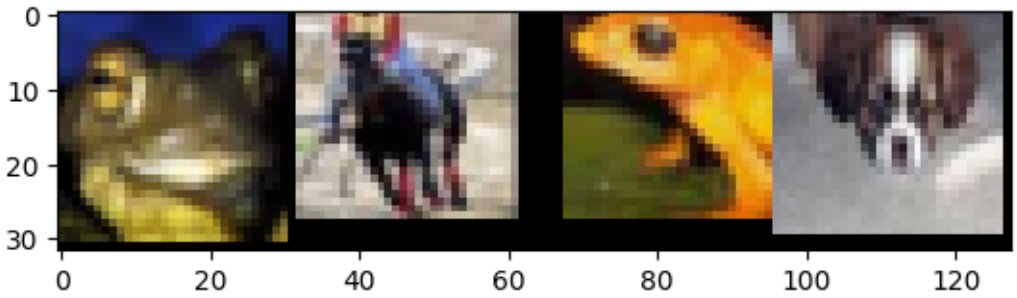
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz

  0%|          | 0/170498071 [00:00<?, ?it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

1.0.1 Show some example images and rotated images with labels:

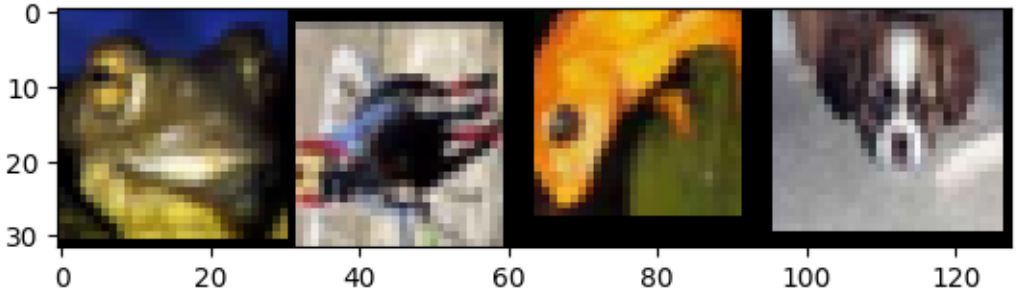
[3]:
```
import matplotlib.pyplot as plt

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

rot_classes = ('0', '90', '180', '270')


def imshow(img):
    # unnormalize
    img = transforms.Normalize((0, 0, 0), (1/0.2023, 1/0.1994, 1/0.2010))(img)
    img = transforms.Normalize((-0.4914, -0.4822, -0.4465), (1, 1, 1))(img)
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


dataiter = iter(trainloader)
images, rot_images, rot_labels, labels = next(dataiter)

# print images and rotated images
img_grid = imshow(torchvision.utils.make_grid(images[:4], padding=0))
print('Class labels: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))
img_grid = imshow(torchvision.utils.make_grid(rot_images[:4], padding=0))
```

```
print('Rotation labels: ', ' '.join(f'{rot_classes[rot_labels[j]]:5s}' for j in␣
 ↪range(4)))
```

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




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Class labels:  frog  horse frog  dog
```




```
Rotation labels:  0     90    90    0
```

2 Evaluation code

[4]:
```
import time

def run_test(net, testloader, criterion, task):
    correct = 0
    total = 0
    avg_test_loss = 0.0
    # since we're not training, we don't need to calculate the gradients for␣
 ↪our outputs
```

```
    with torch.no_grad():
        for images, images_rotated, labels, cls_labels in testloader:
            if task == 'rotation':
                images, labels = images_rotated.to(device), labels.to(device)
            elif task == 'classification':
                images, labels = images.to(device), cls_labels.to(device)
            # TODO: Calculate outputs by running images through the network
            # The class with the highest energy is what we choose as prediction
            outputs = net(images)
            _, predicted = torch.max(outputs, 1)
            correct = correct + (predicted == labels).sum().item()
            total = total + labels.size(0)
            # loss
            avg_test_loss += criterion(outputs, labels) / len(testloader)
    print('TESTING:')
    print(f'Accuracy of the network on the 10000 test images: {100 * correct /␣
↪total:.2f} %')
    print(f'Average loss on the 10000 test images: {avg_test_loss:.3f}')
```

[5]:
```
def adjust_learning_rate(optimizer, epoch, init_lr, decay_epochs=30):
    """Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
    lr = init_lr * (0.1 ** (epoch // decay_epochs))
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
```

3 Train a ResNet18 on the rotation task

3.0.1 In this section, we will train a ResNet18 model on the rotation task. The input is a rotated image and the model predicts the rotation label. See the Data Setup section for details.

[6]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

[6]:
```
'cuda'
```

[7]:
```
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet18

net = resnet18(num_classes=4)
net = net.to(device)
```

[8]:
```
import torch.optim as optim
criterion = None
optimizer = None
```

```
# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[9]:
```
# Both the self-supervised rotation task and supervised CIFAR10 classification␣
 ↪are
# trained with the CrossEntropyLoss, so we can use the training loop code.

def train(net, criterion, optimizer, num_epochs, decay_epochs, init_lr, task):

    for epoch in range(num_epochs):  # loop over the dataset multiple times

        running_loss = 0.0
        running_correct = 0.0
        running_total = 0.0
        start_time = time.time()

        net.train()

        for i, (imgs, imgs_rotated, rotation_label, cls_label) in␣
 ↪enumerate(trainloader, 0):
            adjust_learning_rate(optimizer, epoch, init_lr, decay_epochs)

            # TODO: Set the data to the correct device; Different task will use␣
 ↪different inputs and labels
            if task == 'rotation':
                inputs = imgs_rotated.to(device)
                labels = rotation_label.to(device)
            elif task == 'classification':
                inputs = imgs.to(device)
                labels = cls_label.to(device)
            else:
                raise ValueError('invalid task name')

            # TODO: Zero the parameter gradients
            optimizer.zero_grad()

            # TODO: forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # TODO: Get predicted results
            _, predicted = torch.max(outputs, 1)
```

```
            # print statistics
            print_freq = 100
            running_loss += loss.item()

            # calc acc
            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()

            if i % print_freq == (print_freq - 1):    # print every 2000␣
 ↪mini-batches
                print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss /␣
 ↪print_freq:.3f} acc: {100*running_correct / running_total:.2f} time: {time.
 ↪time() - start_time:.2f}')
                running_loss, running_correct, running_total = 0.0, 0.0, 0.0
                start_time = time.time()

        # TODO: Run the run_test() function after each epoch; Set the model to␣
 ↪the evaluation mode.
        net.eval()
        run_test(net, testloader, criterion, task)

    print('Finished Training')
```

[10]:
```
train(net, criterion, optimizer, num_epochs=45, decay_epochs=15, init_lr=0.01,␣
 ↪task='rotation')

# TODO: Save the model
PATH = './model.pth'
torch.save(net.state_dict(), PATH)
```

```
[1,   100] loss: 1.358 acc: 39.19 time: 22.77
[1,   200] loss: 1.178 acc: 47.58 time: 12.96
[1,   300] loss: 1.160 acc: 49.27 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 51.11 %
Average loss on the 10000 test images: 1.151
[2,   100] loss: 1.107 acc: 51.95 time: 13.09
[2,   200] loss: 1.082 acc: 53.75 time: 12.99
[2,   300] loss: 1.065 acc: 55.02 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 57.73 %
Average loss on the 10000 test images: 1.009
[3,   100] loss: 1.030 acc: 56.60 time: 13.13
[3,   200] loss: 1.012 acc: 57.20 time: 12.94
[3,   300] loss: 1.010 acc: 57.36 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 59.63 %
```

```
Average loss on the 10000 test images: 0.955
[4,   100] loss: 1.007 acc: 57.88 time: 13.08
[4,   200] loss: 0.984 acc: 58.47 time: 12.91
[4,   300] loss: 0.966 acc: 59.70 time: 12.93
TESTING:
Accuracy of the network on the 10000 test images: 60.52 %
Average loss on the 10000 test images: 0.954
[5,   100] loss: 0.939 acc: 61.06 time: 13.07
[5,   200] loss: 0.962 acc: 59.87 time: 12.94
[5,   300] loss: 0.933 acc: 60.80 time: 12.91
TESTING:
Accuracy of the network on the 10000 test images: 63.50 %
Average loss on the 10000 test images: 0.874
[6,   100] loss: 0.922 acc: 61.68 time: 13.18
[6,   200] loss: 0.914 acc: 61.84 time: 12.96
[6,   300] loss: 0.902 acc: 62.27 time: 12.93
TESTING:
Accuracy of the network on the 10000 test images: 63.70 %
Average loss on the 10000 test images: 0.875
[7,   100] loss: 0.898 acc: 63.27 time: 13.17
[7,   200] loss: 0.884 acc: 63.55 time: 12.95
[7,   300] loss: 0.898 acc: 62.98 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 64.59 %
Average loss on the 10000 test images: 0.844
[8,   100] loss: 0.868 acc: 64.39 time: 13.15
[8,   200] loss: 0.866 acc: 64.29 time: 13.00
[8,   300] loss: 0.866 acc: 64.34 time: 13.01
TESTING:
Accuracy of the network on the 10000 test images: 63.61 %
Average loss on the 10000 test images: 0.877
[9,   100] loss: 0.867 acc: 64.01 time: 13.13
[9,   200] loss: 0.867 acc: 63.95 time: 13.01
[9,   300] loss: 0.850 acc: 64.68 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 66.73 %
Average loss on the 10000 test images: 0.810
[10,   100] loss: 0.838 acc: 65.49 time: 13.14
[10,   200] loss: 0.852 acc: 64.96 time: 12.96
[10,   300] loss: 0.830 acc: 66.61 time: 13.04
TESTING:
Accuracy of the network on the 10000 test images: 66.77 %
Average loss on the 10000 test images: 0.819
[11,   100] loss: 0.842 acc: 65.09 time: 13.12
[11,   200] loss: 0.822 acc: 66.23 time: 13.00
[11,   300] loss: 0.818 acc: 66.50 time: 13.01
TESTING:
Accuracy of the network on the 10000 test images: 68.11 %
```

```
Average loss on the 10000 test images: 0.782
[12,   100] loss: 0.805 acc: 67.10 time: 13.13
[12,   200] loss: 0.800 acc: 67.21 time: 12.94
[12,   300] loss: 0.819 acc: 66.59 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 68.99 %
Average loss on the 10000 test images: 0.773
[13,   100] loss: 0.796 acc: 67.84 time: 13.13
[13,   200] loss: 0.809 acc: 66.72 time: 13.01
[13,   300] loss: 0.786 acc: 68.22 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 68.20 %
Average loss on the 10000 test images: 0.779
[14,   100] loss: 0.796 acc: 67.67 time: 13.16
[14,   200] loss: 0.777 acc: 68.36 time: 12.94
[14,   300] loss: 0.783 acc: 68.05 time: 12.96
TESTING:
Accuracy of the network on the 10000 test images: 69.34 %
Average loss on the 10000 test images: 0.751
[15,   100] loss: 0.765 acc: 69.02 time: 13.16
[15,   200] loss: 0.789 acc: 68.00 time: 12.97
[15,   300] loss: 0.769 acc: 68.84 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 68.50 %
Average loss on the 10000 test images: 0.787
[16,   100] loss: 0.745 acc: 69.92 time: 13.20
[16,   200] loss: 0.735 acc: 70.30 time: 12.94
[16,   300] loss: 0.721 acc: 71.23 time: 12.98
TESTING:
Accuracy of the network on the 10000 test images: 72.20 %
Average loss on the 10000 test images: 0.691
[17,   100] loss: 0.720 acc: 70.80 time: 13.22
[17,   200] loss: 0.715 acc: 71.36 time: 12.94
[17,   300] loss: 0.714 acc: 70.91 time: 12.98
TESTING:
Accuracy of the network on the 10000 test images: 72.79 %
Average loss on the 10000 test images: 0.680
[18,   100] loss: 0.713 acc: 70.87 time: 13.13
[18,   200] loss: 0.708 acc: 71.59 time: 13.03
[18,   300] loss: 0.707 acc: 71.52 time: 12.96
TESTING:
Accuracy of the network on the 10000 test images: 72.79 %
Average loss on the 10000 test images: 0.685
[19,   100] loss: 0.721 acc: 71.33 time: 13.14
[19,   200] loss: 0.702 acc: 71.73 time: 13.01
[19,   300] loss: 0.706 acc: 71.29 time: 13.01
TESTING:
Accuracy of the network on the 10000 test images: 73.40 %
```

```
Average loss on the 10000 test images: 0.669
[20,   100] loss: 0.707 acc: 71.48 time: 13.12
[20,   200] loss: 0.703 acc: 71.32 time: 12.95
[20,   300] loss: 0.695 acc: 72.22 time: 13.01
TESTING:
Accuracy of the network on the 10000 test images: 73.18 %
Average loss on the 10000 test images: 0.674
[21,   100] loss: 0.687 acc: 72.55 time: 13.17
[21,   200] loss: 0.699 acc: 71.93 time: 13.00
[21,   300] loss: 0.697 acc: 71.86 time: 13.04
TESTING:
Accuracy of the network on the 10000 test images: 73.84 %
Average loss on the 10000 test images: 0.664
[22,   100] loss: 0.692 acc: 72.31 time: 13.13
[22,   200] loss: 0.698 acc: 71.85 time: 13.00
[22,   300] loss: 0.697 acc: 71.93 time: 12.98
TESTING:
Accuracy of the network on the 10000 test images: 73.26 %
Average loss on the 10000 test images: 0.669
[23,   100] loss: 0.683 acc: 72.62 time: 13.14
[23,   200] loss: 0.694 acc: 71.95 time: 12.97
[23,   300] loss: 0.685 acc: 72.97 time: 13.00
TESTING:
Accuracy of the network on the 10000 test images: 73.40 %
Average loss on the 10000 test images: 0.668
[24,   100] loss: 0.688 acc: 72.24 time: 13.12
[24,   200] loss: 0.679 acc: 72.65 time: 12.97
[24,   300] loss: 0.695 acc: 72.25 time: 12.95
TESTING:
Accuracy of the network on the 10000 test images: 73.95 %
Average loss on the 10000 test images: 0.658
[25,   100] loss: 0.700 acc: 72.14 time: 13.12
[25,   200] loss: 0.687 acc: 72.06 time: 13.00
[25,   300] loss: 0.678 acc: 72.69 time: 12.95
TESTING:
Accuracy of the network on the 10000 test images: 73.34 %
Average loss on the 10000 test images: 0.659
[26,   100] loss: 0.682 acc: 72.58 time: 13.17
[26,   200] loss: 0.682 acc: 72.68 time: 12.95
[26,   300] loss: 0.685 acc: 72.55 time: 12.96
TESTING:
Accuracy of the network on the 10000 test images: 73.73 %
Average loss on the 10000 test images: 0.654
[27,   100] loss: 0.684 acc: 72.72 time: 13.16
[27,   200] loss: 0.681 acc: 72.57 time: 12.92
[27,   300] loss: 0.678 acc: 72.60 time: 12.95
TESTING:
Accuracy of the network on the 10000 test images: 73.76 %
```

```
Average loss on the 10000 test images: 0.650
[28,   100] loss: 0.680 acc: 72.94 time: 13.10
[28,   200] loss: 0.687 acc: 72.11 time: 12.99
[28,   300] loss: 0.673 acc: 72.88 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 73.89 %
Average loss on the 10000 test images: 0.656
[29,   100] loss: 0.681 acc: 72.65 time: 13.14
[29,   200] loss: 0.683 acc: 72.43 time: 13.00
[29,   300] loss: 0.671 acc: 73.31 time: 12.98
TESTING:
Accuracy of the network on the 10000 test images: 74.35 %
Average loss on the 10000 test images: 0.645
[30,   100] loss: 0.662 acc: 73.59 time: 13.16
[30,   200] loss: 0.674 acc: 72.70 time: 12.97
[30,   300] loss: 0.671 acc: 73.05 time: 12.98
TESTING:
Accuracy of the network on the 10000 test images: 73.70 %
Average loss on the 10000 test images: 0.655
[31,   100] loss: 0.674 acc: 72.91 time: 13.13
[31,   200] loss: 0.668 acc: 73.02 time: 13.01
[31,   300] loss: 0.680 acc: 73.04 time: 13.00
TESTING:
Accuracy of the network on the 10000 test images: 74.37 %
Average loss on the 10000 test images: 0.640
[32,   100] loss: 0.673 acc: 73.12 time: 13.15
[32,   200] loss: 0.666 acc: 73.73 time: 12.96
[32,   300] loss: 0.668 acc: 72.97 time: 12.96
TESTING:
Accuracy of the network on the 10000 test images: 74.57 %
Average loss on the 10000 test images: 0.641
[33,   100] loss: 0.664 acc: 73.59 time: 13.10
[33,   200] loss: 0.673 acc: 73.44 time: 12.98
[33,   300] loss: 0.670 acc: 73.14 time: 13.02
TESTING:
Accuracy of the network on the 10000 test images: 74.66 %
Average loss on the 10000 test images: 0.640
[34,   100] loss: 0.661 acc: 73.71 time: 13.12
[34,   200] loss: 0.664 acc: 73.61 time: 12.99
[34,   300] loss: 0.675 acc: 72.95 time: 12.95
TESTING:
Accuracy of the network on the 10000 test images: 74.57 %
Average loss on the 10000 test images: 0.645
[35,   100] loss: 0.659 acc: 73.90 time: 13.17
[35,   200] loss: 0.665 acc: 73.40 time: 12.94
[35,   300] loss: 0.672 acc: 73.02 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 74.27 %
```

```
Average loss on the 10000 test images: 0.644
[36,   100] loss: 0.680 acc: 72.84 time: 13.20
[36,   200] loss: 0.668 acc: 73.28 time: 12.95
[36,   300] loss: 0.662 acc: 73.90 time: 12.95
TESTING:
Accuracy of the network on the 10000 test images: 74.45 %
Average loss on the 10000 test images: 0.643
[37,   100] loss: 0.666 acc: 73.31 time: 13.21
[37,   200] loss: 0.662 acc: 73.72 time: 12.97
[37,   300] loss: 0.663 acc: 73.72 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 74.74 %
Average loss on the 10000 test images: 0.641
[38,   100] loss: 0.656 acc: 74.10 time: 13.13
[38,   200] loss: 0.666 acc: 73.27 time: 13.00
[38,   300] loss: 0.667 acc: 73.13 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 74.85 %
Average loss on the 10000 test images: 0.635
[39,   100] loss: 0.666 acc: 72.88 time: 13.12
[39,   200] loss: 0.657 acc: 73.63 time: 13.04
[39,   300] loss: 0.663 acc: 73.49 time: 12.94
TESTING:
Accuracy of the network on the 10000 test images: 74.74 %
Average loss on the 10000 test images: 0.639
[40,   100] loss: 0.660 acc: 73.25 time: 13.13
[40,   200] loss: 0.671 acc: 73.16 time: 12.96
[40,   300] loss: 0.667 acc: 73.56 time: 13.03
TESTING:
Accuracy of the network on the 10000 test images: 74.48 %
Average loss on the 10000 test images: 0.640
[41,   100] loss: 0.673 acc: 72.80 time: 13.09
[41,   200] loss: 0.669 acc: 73.71 time: 12.93
[41,   300] loss: 0.648 acc: 74.04 time: 13.00
TESTING:
Accuracy of the network on the 10000 test images: 74.80 %
Average loss on the 10000 test images: 0.639
[42,   100] loss: 0.659 acc: 73.46 time: 13.15
[42,   200] loss: 0.662 acc: 73.77 time: 12.99
[42,   300] loss: 0.658 acc: 73.69 time: 13.02
TESTING:
Accuracy of the network on the 10000 test images: 74.78 %
Average loss on the 10000 test images: 0.644
[43,   100] loss: 0.658 acc: 73.83 time: 13.17
[43,   200] loss: 0.671 acc: 73.17 time: 12.99
[43,   300] loss: 0.661 acc: 73.38 time: 13.01
TESTING:
Accuracy of the network on the 10000 test images: 74.59 %
```

```
Average loss on the 10000 test images: 0.640
[44,   100] loss: 0.658 acc: 73.99 time: 13.12
[44,   200] loss: 0.654 acc: 74.08 time: 13.00
[44,   300] loss: 0.661 acc: 73.56 time: 12.97
TESTING:
Accuracy of the network on the 10000 test images: 74.76 %
Average loss on the 10000 test images: 0.639
[45,   100] loss: 0.668 acc: 73.60 time: 13.27
[45,   200] loss: 0.659 acc: 73.50 time: 13.01
[45,   300] loss: 0.667 acc: 73.38 time: 12.98
TESTING:
Accuracy of the network on the 10000 test images: 74.81 %
Average loss on the 10000 test images: 0.635
Finished Training
```

4 Fine-tuning on the pre-trained model

4.0.1 In this section, we will load the pre-trained ResNet18 model and fine-tune on the classification task. We will freeze all previous layers except for the 'layer4' block and 'fc' layer.

[11]:
```python
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet18

# TODO: Load the pre-trained ResNet18 model
net = resnet18(weights=True).to(device)
# print(net)
print([n for n, _ in net.named_children()])
```

```
/opt/conda/lib/python3.7/site-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and will be removed in 0.15. The current behavior is
equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
/home/liangwiki04/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth

  0%|          | 0.00/44.7M [00:00<?, ?B/s]

['conv1', 'bn1', 'relu', 'maxpool', 'layer1', 'layer2', 'layer3', 'layer4',
'avgpool', 'fc']
```

[12]:
```python
# TODO: Freeze all previous layers; only keep the 'layer4' block and 'fc' layer␣
 ↪trainable
for param in net.parameters():
    param.requires_grad = False
```

```
for param in net.layer4.parameters():
    param.requires_grad = True

for param in net.fc.parameters():
    param.requires_grad = True
```

[13]:
```
# Print all the trainable parameters
params_to_update = net.parameters()
print("Params to learn:")
params_to_update = []
for name,param in net.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)
        print("\t",name)
```

```
Params to learn:
         layer4.0.conv1.weight
         layer4.0.bn1.weight
         layer4.0.bn1.bias
         layer4.0.conv2.weight
         layer4.0.bn2.weight
         layer4.0.bn2.bias
         layer4.0.downsample.0.weight
         layer4.0.downsample.1.weight
         layer4.0.downsample.1.bias
         layer4.1.conv1.weight
         layer4.1.bn1.weight
         layer4.1.bn1.bias
         layer4.1.conv2.weight
         layer4.1.bn2.weight
         layer4.1.bn2.bias
         fc.weight
         fc.bias
```

[14]:
```
# TODO: Define criterion and optimizer
# Note that your optimizer only needs to update the parameters that are␣
 ↪trainable.
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[15]:
```
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = '1'

train(net, criterion, optimizer, num_epochs=20, decay_epochs=10, init_lr=0.01,␣
 ↪task='classification')
```

```
[1,   100] loss: 2.191 acc: 36.92 time: 11.63
```

```
[1,   200] loss: 1.403 acc: 51.34 time: 11.42
[1,   300] loss: 1.291 acc: 54.84 time: 11.42
TESTING:
Accuracy of the network on the 10000 test images: 58.26 %
Average loss on the 10000 test images: 1.314
[2,   100] loss: 1.168 acc: 58.95 time: 11.58
[2,   200] loss: 1.128 acc: 60.70 time: 11.41
[2,   300] loss: 1.121 acc: 61.08 time: 11.37
TESTING:
Accuracy of the network on the 10000 test images: 62.41 %
Average loss on the 10000 test images: 1.379
[3,   100] loss: 1.057 acc: 62.87 time: 11.69
[3,   200] loss: 1.061 acc: 62.35 time: 11.45
[3,   300] loss: 1.041 acc: 63.70 time: 11.46
TESTING:
Accuracy of the network on the 10000 test images: 61.86 %
Average loss on the 10000 test images: 1.373
[4,   100] loss: 0.997 acc: 64.89 time: 11.63
[4,   200] loss: 1.002 acc: 65.48 time: 11.51
[4,   300] loss: 0.995 acc: 65.23 time: 11.45
TESTING:
Accuracy of the network on the 10000 test images: 64.34 %
Average loss on the 10000 test images: 1.272
[5,   100] loss: 0.954 acc: 66.31 time: 11.64
[5,   200] loss: 0.959 acc: 66.99 time: 11.45
[5,   300] loss: 0.967 acc: 66.06 time: 11.50
TESTING:
Accuracy of the network on the 10000 test images: 63.61 %
Average loss on the 10000 test images: 1.289
[6,   100] loss: 0.938 acc: 67.28 time: 11.67
[6,   200] loss: 0.930 acc: 67.30 time: 11.45
[6,   300] loss: 0.960 acc: 66.70 time: 11.43
TESTING:
Accuracy of the network on the 10000 test images: 64.88 %
Average loss on the 10000 test images: 1.325
[7,   100] loss: 0.919 acc: 67.84 time: 11.60
[7,   200] loss: 0.923 acc: 67.91 time: 11.44
[7,   300] loss: 0.929 acc: 67.44 time: 11.44
TESTING:
Accuracy of the network on the 10000 test images: 66.75 %
Average loss on the 10000 test images: 1.156
[8,   100] loss: 0.887 acc: 68.11 time: 11.71
[8,   200] loss: 0.905 acc: 68.06 time: 11.44
[8,   300] loss: 0.887 acc: 68.88 time: 11.45
TESTING:
Accuracy of the network on the 10000 test images: 64.97 %
Average loss on the 10000 test images: 1.261
[9,   100] loss: 0.867 acc: 69.37 time: 11.59
```

```
[9,   200] loss: 0.864 acc: 69.38 time: 11.51
[9,   300] loss: 0.881 acc: 68.97 time: 11.50
TESTING:
Accuracy of the network on the 10000 test images: 65.47 %
Average loss on the 10000 test images: 1.290
[10,   100] loss: 0.844 acc: 69.88 time: 11.67
[10,   200] loss: 0.858 acc: 69.63 time: 11.45
[10,   300] loss: 0.863 acc: 69.71 time: 11.50
TESTING:
Accuracy of the network on the 10000 test images: 68.68 %
Average loss on the 10000 test images: 0.945
[11,   100] loss: 0.792 acc: 72.17 time: 11.66
[11,   200] loss: 0.800 acc: 71.77 time: 11.44
[11,   300] loss: 0.788 acc: 72.20 time: 11.44
TESTING:
Accuracy of the network on the 10000 test images: 69.63 %
Average loss on the 10000 test images: 0.906
[12,   100] loss: 0.760 acc: 73.33 time: 11.62
[12,   200] loss: 0.779 acc: 72.35 time: 11.49
[12,   300] loss: 0.758 acc: 72.98 time: 11.48
TESTING:
Accuracy of the network on the 10000 test images: 68.25 %
Average loss on the 10000 test images: 1.027
[13,   100] loss: 0.754 acc: 73.26 time: 11.69
[13,   200] loss: 0.766 acc: 73.01 time: 11.45
[13,   300] loss: 0.785 acc: 72.29 time: 11.47
TESTING:
Accuracy of the network on the 10000 test images: 70.53 %
Average loss on the 10000 test images: 0.862
[14,   100] loss: 0.747 acc: 73.76 time: 11.66
[14,   200] loss: 0.768 acc: 72.67 time: 11.56
[14,   300] loss: 0.750 acc: 73.58 time: 11.49
TESTING:
Accuracy of the network on the 10000 test images: 70.00 %
Average loss on the 10000 test images: 0.884
[15,   100] loss: 0.765 acc: 73.04 time: 11.65
[15,   200] loss: 0.757 acc: 73.16 time: 11.44
[15,   300] loss: 0.758 acc: 73.02 time: 11.51
TESTING:
Accuracy of the network on the 10000 test images: 69.83 %
Average loss on the 10000 test images: 0.924
[16,   100] loss: 0.740 acc: 73.78 time: 11.65
[16,   200] loss: 0.738 acc: 74.16 time: 11.45
[16,   300] loss: 0.759 acc: 72.70 time: 11.46
TESTING:
Accuracy of the network on the 10000 test images: 70.80 %
Average loss on the 10000 test images: 0.861
[17,   100] loss: 0.749 acc: 73.20 time: 11.68
```

```
[17,   200] loss: 0.748 acc: 73.28 time: 11.50
[17,   300] loss: 0.729 acc: 73.87 time: 11.55
TESTING:
Accuracy of the network on the 10000 test images: 69.91 %
Average loss on the 10000 test images: 0.898
[18,   100] loss: 0.754 acc: 73.05 time: 11.76
[18,   200] loss: 0.740 acc: 73.81 time: 11.48
[18,   300] loss: 0.729 acc: 74.64 time: 11.52
TESTING:
Accuracy of the network on the 10000 test images: 70.79 %
Average loss on the 10000 test images: 0.859
[19,   100] loss: 0.742 acc: 74.06 time: 11.63
[19,   200] loss: 0.748 acc: 73.23 time: 11.55
[19,   300] loss: 0.738 acc: 73.95 time: 11.50
TESTING:
Accuracy of the network on the 10000 test images: 70.65 %
Average loss on the 10000 test images: 0.881
[20,   100] loss: 0.724 acc: 74.68 time: 11.71
[20,   200] loss: 0.720 acc: 74.30 time: 11.48
[20,   300] loss: 0.749 acc: 73.27 time: 11.52
TESTING:
Accuracy of the network on the 10000 test images: 69.37 %
Average loss on the 10000 test images: 0.997
Finished Training
```

5 Fine-tuning on the randomly initialized model

5.0.1 In this section, we will randomly initialize a ResNet18 model and fine-tune on the classification task. We will freeze all previous layers except for the 'layer4' block and 'fc' layer.

[16]:
```python
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet18

# TODO: Randomly initialize a ResNet18 model
net = resnet18().to(device)
# print(net)
print([n for n, _ in net.named_children()])
```

```
['conv1', 'bn1', 'relu', 'maxpool', 'layer1', 'layer2', 'layer3', 'layer4',
'avgpool', 'fc']
```

[17]:
```python
# TODO: Freeze all previous layers; only keep the 'layer4' block and 'fc' layer␣
 ↪trainable
# To do this, you should set requires_grad=False for the frozen layers.
for param in net.parameters():
```

```
    param.requires_grad = False

for param in net.layer4.parameters():
    param.requires_grad = True

for param in net.fc.parameters():
    param.requires_grad = True
```

[18]:
```
# Print all the trainable parameters
params_to_update = net.parameters()
print("Params to learn:")
params_to_update = []
for name,param in net.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)
        print("\t",name)
```

```
Params to learn:
         layer4.0.conv1.weight
         layer4.0.bn1.weight
         layer4.0.bn1.bias
         layer4.0.conv2.weight
         layer4.0.bn2.weight
         layer4.0.bn2.bias
         layer4.0.downsample.0.weight
         layer4.0.downsample.1.weight
         layer4.0.downsample.1.bias
         layer4.1.conv1.weight
         layer4.1.bn1.weight
         layer4.1.bn1.bias
         layer4.1.conv2.weight
         layer4.1.bn2.weight
         layer4.1.bn2.bias
         fc.weight
         fc.bias
```

[19]:
```
# TODO: Define criterion and optimizer
# Note that your optimizer only needs to update the parameters that are␣
 ↪trainable.
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[20]:
```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=10, init_lr=0.01,␣
 ↪task='classification')
```

```
[1,   100] loss: 2.356 acc: 23.04 time: 11.58
[1,   200] loss: 1.990 acc: 28.62 time: 11.36
[1,   300] loss: 1.950 acc: 29.88 time: 11.37
```

```
TESTING:
Accuracy of the network on the 10000 test images: 33.57 %
Average loss on the 10000 test images: 1.908
[2,   100] loss: 1.891 acc: 32.14 time: 11.67
[2,   200] loss: 1.864 acc: 32.96 time: 11.36
[2,   300] loss: 1.838 acc: 33.54 time: 11.38
TESTING:
Accuracy of the network on the 10000 test images: 37.61 %
Average loss on the 10000 test images: 1.726
[3,   100] loss: 1.842 acc: 33.37 time: 11.58
[3,   200] loss: 1.813 acc: 35.20 time: 11.41
[3,   300] loss: 1.804 acc: 34.30 time: 11.34
TESTING:
Accuracy of the network on the 10000 test images: 38.84 %
Average loss on the 10000 test images: 1.734
[4,   100] loss: 1.789 acc: 35.38 time: 11.54
[4,   200] loss: 1.768 acc: 36.01 time: 11.34
[4,   300] loss: 1.762 acc: 36.30 time: 11.41
TESTING:
Accuracy of the network on the 10000 test images: 39.66 %
Average loss on the 10000 test images: 1.679
[5,   100] loss: 1.768 acc: 36.50 time: 11.63
[5,   200] loss: 1.757 acc: 36.55 time: 11.41
[5,   300] loss: 1.764 acc: 36.09 time: 11.38
TESTING:
Accuracy of the network on the 10000 test images: 40.61 %
Average loss on the 10000 test images: 1.665
[6,   100] loss: 1.742 acc: 37.84 time: 11.57
[6,   200] loss: 1.724 acc: 38.39 time: 11.38
[6,   300] loss: 1.747 acc: 36.73 time: 11.43
TESTING:
Accuracy of the network on the 10000 test images: 41.88 %
Average loss on the 10000 test images: 1.626
[7,   100] loss: 1.713 acc: 38.37 time: 11.67
[7,   200] loss: 1.712 acc: 39.26 time: 11.39
[7,   300] loss: 1.708 acc: 38.83 time: 11.40
TESTING:
Accuracy of the network on the 10000 test images: 41.90 %
Average loss on the 10000 test images: 1.613
[8,   100] loss: 1.716 acc: 38.61 time: 11.60
[8,   200] loss: 1.695 acc: 39.16 time: 11.46
[8,   300] loss: 1.711 acc: 38.27 time: 11.39
TESTING:
Accuracy of the network on the 10000 test images: 41.30 %
Average loss on the 10000 test images: 1.645
[9,   100] loss: 1.693 acc: 39.26 time: 11.58
[9,   200] loss: 1.676 acc: 40.42 time: 11.43
[9,   300] loss: 1.691 acc: 38.59 time: 11.48
```

```
TESTING:
Accuracy of the network on the 10000 test images: 43.44 %
Average loss on the 10000 test images: 1.586
[10,   100] loss: 1.664 acc: 40.12 time: 11.59
[10,   200] loss: 1.683 acc: 39.62 time: 11.41
[10,   300] loss: 1.682 acc: 39.00 time: 11.41
TESTING:
Accuracy of the network on the 10000 test images: 42.85 %
Average loss on the 10000 test images: 1.599
[11,   100] loss: 1.632 acc: 41.74 time: 11.59
[11,   200] loss: 1.621 acc: 42.14 time: 11.37
[11,   300] loss: 1.613 acc: 41.77 time: 11.39
TESTING:
Accuracy of the network on the 10000 test images: 44.93 %
Average loss on the 10000 test images: 1.553
[12,   100] loss: 1.605 acc: 43.10 time: 11.66
[12,   200] loss: 1.609 acc: 42.01 time: 11.39
[12,   300] loss: 1.616 acc: 42.01 time: 11.38
TESTING:
Accuracy of the network on the 10000 test images: 44.88 %
Average loss on the 10000 test images: 1.556
[13,   100] loss: 1.604 acc: 41.94 time: 11.57
[13,   200] loss: 1.597 acc: 43.08 time: 11.45
[13,   300] loss: 1.586 acc: 43.05 time: 11.41
TESTING:
Accuracy of the network on the 10000 test images: 44.92 %
Average loss on the 10000 test images: 1.545
[14,   100] loss: 1.587 acc: 42.77 time: 11.60
[14,   200] loss: 1.594 acc: 42.70 time: 11.44
[14,   300] loss: 1.584 acc: 43.32 time: 11.49
TESTING:
Accuracy of the network on the 10000 test images: 45.42 %
Average loss on the 10000 test images: 1.540
[15,   100] loss: 1.589 acc: 42.76 time: 11.57
[15,   200] loss: 1.584 acc: 43.11 time: 11.39
[15,   300] loss: 1.592 acc: 43.14 time: 11.38
TESTING:
Accuracy of the network on the 10000 test images: 45.38 %
Average loss on the 10000 test images: 1.544
[16,   100] loss: 1.591 acc: 43.23 time: 11.55
[16,   200] loss: 1.583 acc: 43.21 time: 11.38
[16,   300] loss: 1.603 acc: 42.23 time: 11.40
TESTING:
Accuracy of the network on the 10000 test images: 45.36 %
Average loss on the 10000 test images: 1.537
[17,   100] loss: 1.584 acc: 43.18 time: 11.65
[17,   200] loss: 1.582 acc: 43.73 time: 11.39
[17,   300] loss: 1.588 acc: 42.80 time: 11.42
```

```
TESTING:
Accuracy of the network on the 10000 test images: 45.80 %
Average loss on the 10000 test images: 1.535
[18,   100] loss: 1.579 acc: 43.38 time: 11.58
[18,   200] loss: 1.562 acc: 44.11 time: 11.44
[18,   300] loss: 1.585 acc: 43.58 time: 11.39
TESTING:
Accuracy of the network on the 10000 test images: 45.59 %
Average loss on the 10000 test images: 1.531
[19,   100] loss: 1.568 acc: 43.96 time: 11.56
[19,   200] loss: 1.576 acc: 43.71 time: 11.40
[19,   300] loss: 1.588 acc: 42.93 time: 11.45
TESTING:
Accuracy of the network on the 10000 test images: 45.14 %
Average loss on the 10000 test images: 1.536
[20,   100] loss: 1.564 acc: 44.23 time: 11.57
[20,   200] loss: 1.574 acc: 43.78 time: 11.46
[20,   300] loss: 1.595 acc: 42.73 time: 11.38
TESTING:
Accuracy of the network on the 10000 test images: 45.65 %
Average loss on the 10000 test images: 1.537
Finished Training
```

6 Supervised training on the pre-trained model

6.0.1 In this section, we will load the pre-trained ResNet18 model and re-train the whole model on the classification task.

[21]:
```
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet18

# TODO: Load the pre-trained ResNet18 model
net = resnet18(weights=True).to(device)
for param in net.parameters():
    param.requires_grad = True
```

[22]:
```
# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[23]:
```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=10, init_lr=0.01,␣
 ↪task='classification')
```

```
[1,   100] loss: 2.302 acc: 26.05 time: 13.26
[1,   200] loss: 1.521 acc: 45.63 time: 13.06
[1,   300] loss: 1.185 acc: 59.16 time: 13.04
```

```
TESTING:
Accuracy of the network on the 10000 test images: 68.38 %
Average loss on the 10000 test images: 0.915
[2,   100] loss: 0.924 acc: 68.23 time: 13.26
[2,   200] loss: 0.882 acc: 70.16 time: 13.13
[2,   300] loss: 0.827 acc: 72.02 time: 13.13
TESTING:
Accuracy of the network on the 10000 test images: 72.96 %
Average loss on the 10000 test images: 0.823
[3,   100] loss: 0.749 acc: 74.80 time: 13.38
[3,   200] loss: 0.721 acc: 75.64 time: 13.16
[3,   300] loss: 0.685 acc: 76.59 time: 13.15
TESTING:
Accuracy of the network on the 10000 test images: 76.86 %
Average loss on the 10000 test images: 0.689
[4,   100] loss: 0.637 acc: 78.11 time: 13.37
[4,   200] loss: 0.627 acc: 78.26 time: 13.14
[4,   300] loss: 0.630 acc: 78.79 time: 13.12
TESTING:
Accuracy of the network on the 10000 test images: 78.61 %
Average loss on the 10000 test images: 0.626
[5,   100] loss: 0.627 acc: 78.78 time: 13.36
[5,   200] loss: 0.592 acc: 79.67 time: 13.17
[5,   300] loss: 0.577 acc: 80.64 time: 13.16
TESTING:
Accuracy of the network on the 10000 test images: 80.55 %
Average loss on the 10000 test images: 0.565
[6,   100] loss: 0.536 acc: 81.45 time: 13.31
[6,   200] loss: 0.558 acc: 80.60 time: 13.20
[6,   300] loss: 0.554 acc: 81.00 time: 13.15
TESTING:
Accuracy of the network on the 10000 test images: 80.02 %
Average loss on the 10000 test images: 0.584
[7,   100] loss: 0.490 acc: 83.13 time: 13.33
[7,   200] loss: 0.516 acc: 82.19 time: 13.18
[7,   300] loss: 0.517 acc: 82.59 time: 13.14
TESTING:
Accuracy of the network on the 10000 test images: 82.10 %
Average loss on the 10000 test images: 0.543
[8,   100] loss: 0.532 acc: 82.41 time: 13.31
[8,   200] loss: 0.510 acc: 82.28 time: 13.13
[8,   300] loss: 0.490 acc: 83.03 time: 13.16
TESTING:
Accuracy of the network on the 10000 test images: 83.63 %
Average loss on the 10000 test images: 0.491
[9,   100] loss: 0.465 acc: 83.87 time: 13.31
[9,   200] loss: 0.462 acc: 84.14 time: 13.12
[9,   300] loss: 0.460 acc: 84.39 time: 13.20
```

```
TESTING:
Accuracy of the network on the 10000 test images: 83.23 %
Average loss on the 10000 test images: 0.506
[10,   100] loss: 0.440 acc: 84.85 time: 13.29
[10,   200] loss: 0.445 acc: 84.45 time: 13.15
[10,   300] loss: 0.454 acc: 84.40 time: 13.17
TESTING:
Accuracy of the network on the 10000 test images: 83.49 %
Average loss on the 10000 test images: 0.494
[11,   100] loss: 0.375 acc: 87.21 time: 13.29
[11,   200] loss: 0.348 acc: 88.05 time: 13.12
[11,   300] loss: 0.340 acc: 88.35 time: 13.12
TESTING:
Accuracy of the network on the 10000 test images: 85.42 %
Average loss on the 10000 test images: 0.434
[12,   100] loss: 0.331 acc: 88.65 time: 13.29
[12,   200] loss: 0.335 acc: 88.40 time: 13.12
[12,   300] loss: 0.329 acc: 88.77 time: 13.14
TESTING:
Accuracy of the network on the 10000 test images: 85.80 %
Average loss on the 10000 test images: 0.426
[13,   100] loss: 0.313 acc: 89.00 time: 13.30
[13,   200] loss: 0.319 acc: 88.87 time: 13.10
[13,   300] loss: 0.326 acc: 88.89 time: 13.17
TESTING:
Accuracy of the network on the 10000 test images: 85.93 %
Average loss on the 10000 test images: 0.425
[14,   100] loss: 0.315 acc: 89.03 time: 13.33
[14,   200] loss: 0.311 acc: 89.16 time: 13.11
[14,   300] loss: 0.305 acc: 89.30 time: 13.16
TESTING:
Accuracy of the network on the 10000 test images: 86.02 %
Average loss on the 10000 test images: 0.423
[15,   100] loss: 0.302 acc: 89.84 time: 13.34
[15,   200] loss: 0.312 acc: 89.20 time: 13.14
[15,   300] loss: 0.300 acc: 89.48 time: 13.12
TESTING:
Accuracy of the network on the 10000 test images: 85.97 %
Average loss on the 10000 test images: 0.423
[16,   100] loss: 0.299 acc: 89.45 time: 13.36
[16,   200] loss: 0.306 acc: 89.52 time: 13.13
[16,   300] loss: 0.301 acc: 89.54 time: 13.15
TESTING:
Accuracy of the network on the 10000 test images: 86.17 %
Average loss on the 10000 test images: 0.420
[17,   100] loss: 0.294 acc: 89.77 time: 13.31
[17,   200] loss: 0.295 acc: 89.74 time: 13.17
[17,   300] loss: 0.295 acc: 89.90 time: 13.11
```

```
TESTING:
Accuracy of the network on the 10000 test images: 85.93 %
Average loss on the 10000 test images: 0.423
[18,   100] loss: 0.291 acc: 89.80 time: 13.32
[18,   200] loss: 0.292 acc: 89.70 time: 13.24
[18,   300] loss: 0.285 acc: 90.38 time: 13.13
TESTING:
Accuracy of the network on the 10000 test images: 86.27 %
Average loss on the 10000 test images: 0.420
[19,   100] loss: 0.288 acc: 89.92 time: 13.36
[19,   200] loss: 0.274 acc: 90.27 time: 13.24
[19,   300] loss: 0.288 acc: 89.91 time: 13.15
TESTING:
Accuracy of the network on the 10000 test images: 86.18 %
Average loss on the 10000 test images: 0.420
[20,   100] loss: 0.271 acc: 90.53 time: 13.27
[20,   200] loss: 0.274 acc: 90.45 time: 13.12
[20,   300] loss: 0.283 acc: 89.93 time: 13.17
TESTING:
Accuracy of the network on the 10000 test images: 86.17 %
Average loss on the 10000 test images: 0.422
Finished Training
```

7 Supervised training on the randomly initialized model

7.0.1 In this section, we will randomly initialize a ResNet18 model and re-train the whole model on the classification task.

[24]:
```
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet18

# TODO: Randomly initialize a ResNet18 model
net = resnet18().to(device)
for param in net.parameters():
    param.requires_grad = True
```

[25]:
```
# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[26]:
```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=10, init_lr=0.01,␣
 ↪task='classification')
```

```
[1,   100] loss: 2.195 acc: 27.48 time: 13.28
[1,   200] loss: 1.699 acc: 38.18 time: 13.05
[1,   300] loss: 1.581 acc: 41.94 time: 13.05
```

```
TESTING:
Accuracy of the network on the 10000 test images: 49.58 %
Average loss on the 10000 test images: 1.355
[2,   100] loss: 1.426 acc: 48.40 time: 13.30
[2,   200] loss: 1.365 acc: 50.84 time: 13.12
[2,   300] loss: 1.323 acc: 51.94 time: 13.13
TESTING:
Accuracy of the network on the 10000 test images: 54.44 %
Average loss on the 10000 test images: 1.267
[3,   100] loss: 1.227 acc: 55.64 time: 13.27
[3,   200] loss: 1.194 acc: 57.54 time: 13.09
[3,   300] loss: 1.158 acc: 58.85 time: 13.11
TESTING:
Accuracy of the network on the 10000 test images: 63.36 %
Average loss on the 10000 test images: 1.039
[4,   100] loss: 1.059 acc: 62.05 time: 13.29
[4,   200] loss: 1.081 acc: 61.37 time: 13.11
[4,   300] loss: 1.057 acc: 62.58 time: 13.11
TESTING:
Accuracy of the network on the 10000 test images: 63.96 %
Average loss on the 10000 test images: 1.003
[5,   100] loss: 0.997 acc: 64.66 time: 13.38
[5,   200] loss: 0.994 acc: 64.74 time: 13.12
[5,   300] loss: 0.974 acc: 65.38 time: 13.12
TESTING:
Accuracy of the network on the 10000 test images: 66.17 %
Average loss on the 10000 test images: 0.966
[6,   100] loss: 0.928 acc: 67.38 time: 13.34
[6,   200] loss: 0.938 acc: 67.02 time: 13.12
[6,   300] loss: 0.914 acc: 67.59 time: 13.11
TESTING:
Accuracy of the network on the 10000 test images: 69.21 %
Average loss on the 10000 test images: 0.882
[7,   100] loss: 0.865 acc: 69.21 time: 13.35
[7,   200] loss: 0.875 acc: 68.69 time: 13.11
[7,   300] loss: 0.871 acc: 70.01 time: 13.14
TESTING:
Accuracy of the network on the 10000 test images: 70.85 %
Average loss on the 10000 test images: 0.841
[8,   100] loss: 0.834 acc: 70.95 time: 13.35
[8,   200] loss: 0.822 acc: 71.26 time: 13.16
[8,   300] loss: 0.821 acc: 70.92 time: 13.15
TESTING:
Accuracy of the network on the 10000 test images: 69.32 %
Average loss on the 10000 test images: 0.891
[9,   100] loss: 0.776 acc: 72.26 time: 13.27
[9,   200] loss: 0.767 acc: 72.84 time: 13.16
[9,   300] loss: 0.797 acc: 71.89 time: 13.10
```

```
TESTING:
Accuracy of the network on the 10000 test images: 73.67 %
Average loss on the 10000 test images: 0.773
[10,   100] loss: 0.744 acc: 73.62 time: 13.29
[10,   200] loss: 0.758 acc: 73.31 time: 13.12
[10,   300] loss: 0.746 acc: 73.56 time: 13.19
TESTING:
Accuracy of the network on the 10000 test images: 70.62 %
Average loss on the 10000 test images: 0.872
[11,   100] loss: 0.667 acc: 76.29 time: 13.31
[11,   200] loss: 0.644 acc: 77.57 time: 13.14
[11,   300] loss: 0.631 acc: 78.11 time: 13.16
TESTING:
Accuracy of the network on the 10000 test images: 77.37 %
Average loss on the 10000 test images: 0.647
[12,   100] loss: 0.628 acc: 78.14 time: 13.28
[12,   200] loss: 0.590 acc: 79.03 time: 13.09
[12,   300] loss: 0.606 acc: 78.89 time: 13.15
TESTING:
Accuracy of the network on the 10000 test images: 77.29 %
Average loss on the 10000 test images: 0.652
[13,   100] loss: 0.605 acc: 78.49 time: 13.28
[13,   200] loss: 0.597 acc: 78.90 time: 13.11
[13,   300] loss: 0.592 acc: 78.76 time: 13.13
TESTING:
Accuracy of the network on the 10000 test images: 77.55 %
Average loss on the 10000 test images: 0.648
[14,   100] loss: 0.588 acc: 79.50 time: 13.36
[14,   200] loss: 0.603 acc: 78.55 time: 13.15
[14,   300] loss: 0.585 acc: 79.08 time: 13.11
TESTING:
Accuracy of the network on the 10000 test images: 78.14 %
Average loss on the 10000 test images: 0.637
[15,   100] loss: 0.583 acc: 79.51 time: 13.33
[15,   200] loss: 0.584 acc: 79.17 time: 13.15
[15,   300] loss: 0.574 acc: 79.88 time: 13.08
TESTING:
Accuracy of the network on the 10000 test images: 78.09 %
Average loss on the 10000 test images: 0.632
[16,   100] loss: 0.577 acc: 79.41 time: 13.38
[16,   200] loss: 0.557 acc: 80.38 time: 13.10
[16,   300] loss: 0.588 acc: 79.28 time: 13.07
TESTING:
Accuracy of the network on the 10000 test images: 78.29 %
Average loss on the 10000 test images: 0.631
[17,   100] loss: 0.578 acc: 79.91 time: 13.35
[17,   200] loss: 0.563 acc: 80.28 time: 13.12
[17,   300] loss: 0.557 acc: 80.45 time: 13.08
```

```
TESTING:
Accuracy of the network on the 10000 test images: 78.63 %
Average loss on the 10000 test images: 0.626
[18,   100] loss: 0.556 acc: 80.45 time: 13.32
[18,   200] loss: 0.559 acc: 80.26 time: 13.12
[18,   300] loss: 0.562 acc: 80.39 time: 13.13
TESTING:
Accuracy of the network on the 10000 test images: 78.30 %
Average loss on the 10000 test images: 0.627
[19,   100] loss: 0.559 acc: 80.23 time: 13.32
[19,   200] loss: 0.551 acc: 80.67 time: 13.17
[19,   300] loss: 0.544 acc: 80.20 time: 13.18
TESTING:
Accuracy of the network on the 10000 test images: 78.67 %
Average loss on the 10000 test images: 0.625
[20,   100] loss: 0.547 acc: 80.98 time: 13.27
[20,   200] loss: 0.554 acc: 80.54 time: 13.20
[20,   300] loss: 0.547 acc: 80.90 time: 13.12
TESTING:
Accuracy of the network on the 10000 test images: 78.62 %
Average loss on the 10000 test images: 0.621
Finished Training
```

[ ]: