a3_part1_rotation

March 17, 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.

#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 `__get_item__` 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.

[ ]:
```
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:
    return torch.rot90(img, k=1, dims=(1, 2))
  elif rot == 2:
    return torch.rot90(img, k=2, dims=(1, 2))
  elif rot == 3:
    return torch.rot90(img, k=-1, dims=(1, 2))
  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()
```

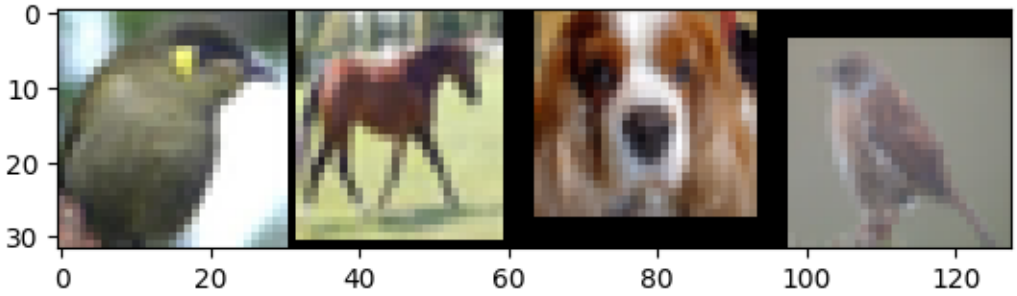
[ ]:
```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

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

100%|      | 170498071/170498071 [00:02<00:00, 82049437.19it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
```

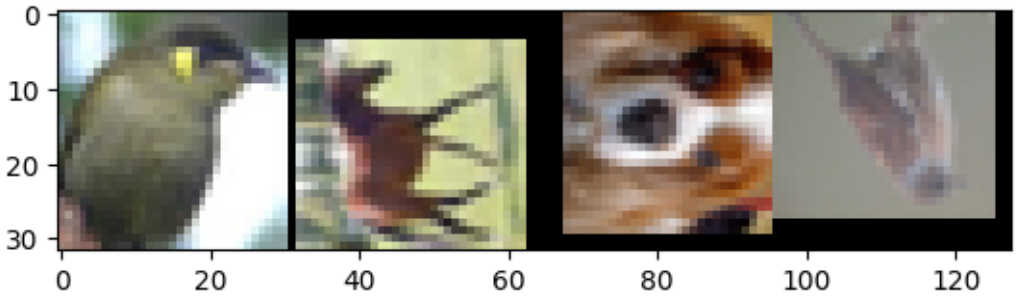
```
Files already downloaded and verified
```

1.0.1 Show some example images and rotated images with labels:

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




```
Class labels:  bird  horse dog   bird
```




```
Rotation labels:  0     90    270   180
```

2 Evaluation code

[ ]:
```python
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
            outputs = net(images)
            # loss
            avg_test_loss += criterion(outputs, labels)  / len(testloader)

            predicted = torch.argmax(outputs, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    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}')
```

[ ]:
```python
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.

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

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

[ ]:
```
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)
```

[ ]:
```
import torch.optim as optim
from tqdm import tqdm

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.002)
```

[ ]:
```
# 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, data 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
            imgs, imgs_rotated, rotation_label, cls_label = data
            imgs, imgs_rotated, rotation_label, cls_label = imgs.to(device),␣
↪imgs_rotated.to(device), rotation_label.to(device), cls_label.to(device)

            # TODO: Zero the parameter gradients
            optimizer.zero_grad()
            outputs = net(imgs_rotated)
            loss = criterion(outputs, rotation_label)
            loss.backward()
            optimizer.step()
            # TODO: Get predicted results
            predicted = torch.argmax(outputs, dim=1)

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

            # calc acc
            running_total += rotation_label.size(0)
            running_correct += (predicted == rotation_label).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.
        run_test(net, testloader, criterion, "rotation")

    print('Finished Training')
```

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

```
[1,   100] loss: 1.481 acc: 35.73 time: 6.11
[1,   200] loss: 1.235 acc: 44.17 time: 4.15
[1,   300] loss: 1.166 acc: 47.92 time: 4.15
TESTING:
Accuracy of the network on the 10000 test images: 51.35 %
Average loss on the 10000 test images: 1.124
[2,   100] loss: 1.143 acc: 49.88 time: 4.39
```

```
[2,   200] loss: 1.110 acc: 52.18 time: 4.35
[2,   300] loss: 1.073 acc: 53.80 time: 4.48
TESTING:
Accuracy of the network on the 10000 test images: 55.89 %
Average loss on the 10000 test images: 1.030
[3,   100] loss: 1.055 acc: 54.79 time: 4.53
[3,   200] loss: 1.035 acc: 56.18 time: 4.31
[3,   300] loss: 1.025 acc: 56.66 time: 4.39
TESTING:
Accuracy of the network on the 10000 test images: 59.01 %
Average loss on the 10000 test images: 0.979
[4,   100] loss: 0.989 acc: 58.52 time: 4.54
[4,   200] loss: 0.997 acc: 57.73 time: 4.40
[4,   300] loss: 0.969 acc: 59.13 time: 4.40
TESTING:
Accuracy of the network on the 10000 test images: 60.45 %
Average loss on the 10000 test images: 0.949
[5,   100] loss: 0.938 acc: 60.81 time: 4.48
[5,   200] loss: 0.944 acc: 60.36 time: 4.38
[5,   300] loss: 0.942 acc: 60.34 time: 4.31
TESTING:
Accuracy of the network on the 10000 test images: 62.77 %
Average loss on the 10000 test images: 0.905
[6,   100] loss: 0.921 acc: 61.48 time: 4.54
[6,   200] loss: 0.912 acc: 62.22 time: 4.52
[6,   300] loss: 0.904 acc: 62.16 time: 4.48
TESTING:
Accuracy of the network on the 10000 test images: 63.73 %
Average loss on the 10000 test images: 0.880
[7,   100] loss: 0.881 acc: 63.13 time: 4.70
[7,   200] loss: 0.892 acc: 62.67 time: 4.52
[7,   300] loss: 0.874 acc: 63.70 time: 4.55
TESTING:
Accuracy of the network on the 10000 test images: 63.62 %
Average loss on the 10000 test images: 0.874
[8,   100] loss: 0.873 acc: 63.98 time: 4.59
[8,   200] loss: 0.863 acc: 64.11 time: 4.55
[8,   300] loss: 0.859 acc: 64.65 time: 4.47
TESTING:
Accuracy of the network on the 10000 test images: 64.90 %
Average loss on the 10000 test images: 0.842
[9,   100] loss: 0.860 acc: 64.45 time: 4.74
[9,   200] loss: 0.847 acc: 64.75 time: 4.56
[9,   300] loss: 0.845 acc: 65.13 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 66.53 %
Average loss on the 10000 test images: 0.816
[10,   100] loss: 0.835 acc: 65.47 time: 4.72
```

```
[10,   200] loss: 0.821 acc: 66.27 time: 4.58
[10,   300] loss: 0.828 acc: 66.10 time: 4.75
TESTING:
Accuracy of the network on the 10000 test images: 67.00 %
Average loss on the 10000 test images: 0.805
[11,   100] loss: 0.800 acc: 67.24 time: 4.75
[11,   200] loss: 0.798 acc: 67.63 time: 4.56
[11,   300] loss: 0.807 acc: 67.03 time: 4.55
TESTING:
Accuracy of the network on the 10000 test images: 68.26 %
Average loss on the 10000 test images: 0.772
[12,   100] loss: 0.795 acc: 67.41 time: 4.72
[12,   200] loss: 0.786 acc: 67.79 time: 4.62
[12,   300] loss: 0.780 acc: 68.38 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 69.27 %
Average loss on the 10000 test images: 0.764
[13,   100] loss: 0.777 acc: 68.47 time: 4.68
[13,   200] loss: 0.767 acc: 69.02 time: 4.57
[13,   300] loss: 0.756 acc: 69.25 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 70.06 %
Average loss on the 10000 test images: 0.735
[14,   100] loss: 0.758 acc: 69.19 time: 4.77
[14,   200] loss: 0.757 acc: 69.61 time: 4.67
[14,   300] loss: 0.736 acc: 70.76 time: 4.53
TESTING:
Accuracy of the network on the 10000 test images: 70.85 %
Average loss on the 10000 test images: 0.721
[15,   100] loss: 0.727 acc: 70.91 time: 4.75
[15,   200] loss: 0.739 acc: 70.20 time: 4.64
[15,   300] loss: 0.729 acc: 70.64 time: 4.59
TESTING:
Accuracy of the network on the 10000 test images: 72.26 %
Average loss on the 10000 test images: 0.694
[16,   100] loss: 0.691 acc: 72.29 time: 4.74
[16,   200] loss: 0.668 acc: 73.98 time: 4.57
[16,   300] loss: 0.664 acc: 73.66 time: 4.62
TESTING:
Accuracy of the network on the 10000 test images: 74.02 %
Average loss on the 10000 test images: 0.654
[17,   100] loss: 0.647 acc: 74.11 time: 4.74
[17,   200] loss: 0.653 acc: 74.07 time: 4.61
[17,   300] loss: 0.656 acc: 74.09 time: 4.57
TESTING:
Accuracy of the network on the 10000 test images: 73.92 %
Average loss on the 10000 test images: 0.646
[18,   100] loss: 0.634 acc: 74.89 time: 4.71
```

```
[18,   200] loss: 0.634 acc: 75.20 time: 4.56
[18,   300] loss: 0.639 acc: 74.52 time: 4.54
TESTING:
Accuracy of the network on the 10000 test images: 74.54 %
Average loss on the 10000 test images: 0.638
[19,   100] loss: 0.624 acc: 75.53 time: 4.77
[19,   200] loss: 0.629 acc: 75.23 time: 4.56
[19,   300] loss: 0.628 acc: 74.83 time: 4.52
TESTING:
Accuracy of the network on the 10000 test images: 74.81 %
Average loss on the 10000 test images: 0.635
[20,   100] loss: 0.626 acc: 75.22 time: 4.77
[20,   200] loss: 0.620 acc: 75.76 time: 4.57
[20,   300] loss: 0.622 acc: 75.48 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 75.20 %
Average loss on the 10000 test images: 0.625
[21,   100] loss: 0.622 acc: 75.84 time: 4.76
[21,   200] loss: 0.608 acc: 75.98 time: 4.55
[21,   300] loss: 0.626 acc: 75.23 time: 4.63
TESTING:
Accuracy of the network on the 10000 test images: 75.82 %
Average loss on the 10000 test images: 0.618
[22,   100] loss: 0.616 acc: 75.89 time: 4.83
[22,   200] loss: 0.619 acc: 75.39 time: 4.63
[22,   300] loss: 0.600 acc: 76.31 time: 4.68
TESTING:
Accuracy of the network on the 10000 test images: 75.65 %
Average loss on the 10000 test images: 0.611
[23,   100] loss: 0.592 acc: 76.49 time: 4.78
[23,   200] loss: 0.614 acc: 75.56 time: 4.61
[23,   300] loss: 0.600 acc: 76.28 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 76.03 %
Average loss on the 10000 test images: 0.613
[24,   100] loss: 0.596 acc: 76.77 time: 4.73
[24,   200] loss: 0.597 acc: 76.29 time: 4.55
[24,   300] loss: 0.592 acc: 76.49 time: 4.60
TESTING:
Accuracy of the network on the 10000 test images: 75.70 %
Average loss on the 10000 test images: 0.611
[25,   100] loss: 0.592 acc: 76.77 time: 4.72
[25,   200] loss: 0.589 acc: 76.68 time: 4.55
[25,   300] loss: 0.594 acc: 76.47 time: 4.53
TESTING:
Accuracy of the network on the 10000 test images: 75.93 %
Average loss on the 10000 test images: 0.608
[26,   100] loss: 0.592 acc: 76.50 time: 4.72
```

```
[26,   200] loss: 0.590 acc: 77.04 time: 4.72
[26,   300] loss: 0.590 acc: 76.87 time: 4.55
TESTING:
Accuracy of the network on the 10000 test images: 76.14 %
Average loss on the 10000 test images: 0.610
[27,   100] loss: 0.591 acc: 76.74 time: 4.83
[27,   200] loss: 0.585 acc: 77.16 time: 4.57
[27,   300] loss: 0.585 acc: 77.17 time: 4.71
TESTING:
Accuracy of the network on the 10000 test images: 76.15 %
Average loss on the 10000 test images: 0.600
[28,   100] loss: 0.580 acc: 77.48 time: 4.76
[28,   200] loss: 0.589 acc: 77.02 time: 4.58
[28,   300] loss: 0.579 acc: 77.06 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 76.69 %
Average loss on the 10000 test images: 0.594
[29,   100] loss: 0.576 acc: 77.09 time: 4.76
[29,   200] loss: 0.572 acc: 77.55 time: 4.69
[29,   300] loss: 0.586 acc: 76.94 time: 4.53
TESTING:
Accuracy of the network on the 10000 test images: 76.87 %
Average loss on the 10000 test images: 0.592
[30,   100] loss: 0.562 acc: 77.77 time: 4.73
[30,   200] loss: 0.578 acc: 77.14 time: 4.60
[30,   300] loss: 0.583 acc: 77.09 time: 4.52
TESTING:
Accuracy of the network on the 10000 test images: 76.77 %
Average loss on the 10000 test images: 0.592
[31,   100] loss: 0.563 acc: 77.85 time: 4.86
[31,   200] loss: 0.556 acc: 78.27 time: 4.55
[31,   300] loss: 0.576 acc: 77.32 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 76.69 %
Average loss on the 10000 test images: 0.590
[32,   100] loss: 0.558 acc: 78.49 time: 4.69
[32,   200] loss: 0.564 acc: 77.88 time: 4.49
[32,   300] loss: 0.564 acc: 77.84 time: 4.62
TESTING:
Accuracy of the network on the 10000 test images: 77.22 %
Average loss on the 10000 test images: 0.585
[33,   100] loss: 0.561 acc: 78.10 time: 4.70
[33,   200] loss: 0.568 acc: 77.89 time: 4.49
[33,   300] loss: 0.564 acc: 78.09 time: 4.48
TESTING:
Accuracy of the network on the 10000 test images: 77.17 %
Average loss on the 10000 test images: 0.583
[34,   100] loss: 0.558 acc: 78.39 time: 4.79
```

```
[34,   200] loss: 0.566 acc: 78.20 time: 4.52
[34,   300] loss: 0.558 acc: 78.26 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 77.30 %
Average loss on the 10000 test images: 0.582
[35,   100] loss: 0.555 acc: 78.12 time: 4.70
[35,   200] loss: 0.565 acc: 77.98 time: 4.58
[35,   300] loss: 0.561 acc: 78.16 time: 4.62
TESTING:
Accuracy of the network on the 10000 test images: 76.89 %
Average loss on the 10000 test images: 0.584
[36,   100] loss: 0.550 acc: 78.90 time: 4.70
[36,   200] loss: 0.557 acc: 78.00 time: 4.47
[36,   300] loss: 0.558 acc: 78.16 time: 4.61
TESTING:
Accuracy of the network on the 10000 test images: 77.07 %
Average loss on the 10000 test images: 0.583
[37,   100] loss: 0.563 acc: 77.86 time: 4.71
[37,   200] loss: 0.553 acc: 78.43 time: 4.68
[37,   300] loss: 0.551 acc: 78.31 time: 4.57
TESTING:
Accuracy of the network on the 10000 test images: 76.91 %
Average loss on the 10000 test images: 0.578
[38,   100] loss: 0.551 acc: 78.28 time: 4.73
[38,   200] loss: 0.553 acc: 78.41 time: 4.53
[38,   300] loss: 0.562 acc: 78.16 time: 4.65
TESTING:
Accuracy of the network on the 10000 test images: 77.22 %
Average loss on the 10000 test images: 0.577
[39,   100] loss: 0.544 acc: 78.97 time: 4.70
[39,   200] loss: 0.560 acc: 77.88 time: 4.52
[39,   300] loss: 0.568 acc: 77.88 time: 4.52
TESTING:
Accuracy of the network on the 10000 test images: 77.12 %
Average loss on the 10000 test images: 0.588
[40,   100] loss: 0.548 acc: 78.49 time: 4.75
[40,   200] loss: 0.557 acc: 78.28 time: 4.56
[40,   300] loss: 0.559 acc: 78.11 time: 4.62
TESTING:
Accuracy of the network on the 10000 test images: 77.50 %
Average loss on the 10000 test images: 0.572
[41,   100] loss: 0.564 acc: 77.91 time: 4.74
[41,   200] loss: 0.559 acc: 78.59 time: 4.57
[41,   300] loss: 0.550 acc: 78.41 time: 4.61
TESTING:
Accuracy of the network on the 10000 test images: 77.28 %
Average loss on the 10000 test images: 0.579
[42,   100] loss: 0.560 acc: 78.29 time: 4.77
```

```
[42,   200] loss: 0.550 acc: 78.84 time: 4.49
[42,   300] loss: 0.554 acc: 78.33 time: 4.53
TESTING:
Accuracy of the network on the 10000 test images: 77.49 %
Average loss on the 10000 test images: 0.578
[43,   100] loss: 0.543 acc: 78.77 time: 4.70
[43,   200] loss: 0.555 acc: 78.09 time: 4.54
[43,   300] loss: 0.559 acc: 78.09 time: 4.55
TESTING:
Accuracy of the network on the 10000 test images: 77.45 %
Average loss on the 10000 test images: 0.578
[44,   100] loss: 0.553 acc: 78.56 time: 4.68
[44,   200] loss: 0.550 acc: 78.23 time: 4.52
[44,   300] loss: 0.548 acc: 78.68 time: 4.53
TESTING:
Accuracy of the network on the 10000 test images: 77.08 %
Average loss on the 10000 test images: 0.580
[45,   100] loss: 0.555 acc: 78.48 time: 4.85
[45,   200] loss: 0.553 acc: 78.22 time: 4.51
[45,   300] loss: 0.542 acc: 78.30 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 77.33 %
Average loss on the 10000 test images: 0.580
Finished Training
```

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

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.

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

from torchvision.models import resnet18

net = resnet18(num_classes=10).to(device)
net_params = torch.load(PATH)
net_params['fc.weight'] = torch.randn(10, net_params['fc.weight'].shape[1])
net_params['fc.bias'] = torch.randn(10)
net.load_state_dict(net_params)
```

[ ]:
```
<All keys matched successfully>
```

[ ]:
```
# 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
```

[ ]:
```
# 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
```

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

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

```
[1,   100] loss: 0.838 acc: 73.67 time: 4.18
```

```
[1,   200] loss: 0.598 acc: 77.02 time: 3.95
[1,   300] loss: 0.583 acc: 77.45 time: 3.94
TESTING:
Accuracy of the network on the 10000 test images: 77.25 %
Average loss on the 10000 test images: 0.584
[2,   100] loss: 0.570 acc: 77.41 time: 3.93
[2,   200] loss: 0.565 acc: 78.09 time: 3.82
[2,   300] loss: 0.557 acc: 78.29 time: 3.84
TESTING:
Accuracy of the network on the 10000 test images: 76.71 %
Average loss on the 10000 test images: 0.594
[3,   100] loss: 0.560 acc: 78.09 time: 4.10
[3,   200] loss: 0.572 acc: 77.91 time: 3.88
[3,   300] loss: 0.560 acc: 78.04 time: 4.04
TESTING:
Accuracy of the network on the 10000 test images: 77.02 %
Average loss on the 10000 test images: 0.584
[4,   100] loss: 0.580 acc: 78.17 time: 4.08
[4,   200] loss: 0.556 acc: 78.24 time: 3.90
[4,   300] loss: 0.557 acc: 78.45 time: 3.97
TESTING:
Accuracy of the network on the 10000 test images: 76.89 %
Average loss on the 10000 test images: 0.584
[5,   100] loss: 0.553 acc: 78.12 time: 4.33
[5,   200] loss: 0.567 acc: 77.77 time: 4.15
[5,   300] loss: 0.565 acc: 77.94 time: 3.86
TESTING:
Accuracy of the network on the 10000 test images: 77.25 %
Average loss on the 10000 test images: 0.584
[6,   100] loss: 0.555 acc: 78.34 time: 3.98
[6,   200] loss: 0.561 acc: 78.07 time: 3.96
[6,   300] loss: 0.548 acc: 79.05 time: 3.93
TESTING:
Accuracy of the network on the 10000 test images: 77.27 %
Average loss on the 10000 test images: 0.581
[7,   100] loss: 0.560 acc: 78.09 time: 4.16
[7,   200] loss: 0.548 acc: 78.72 time: 3.93
[7,   300] loss: 0.561 acc: 78.22 time: 3.77
TESTING:
Accuracy of the network on the 10000 test images: 77.43 %
Average loss on the 10000 test images: 0.580
[8,   100] loss: 0.552 acc: 78.47 time: 4.01
[8,   200] loss: 0.565 acc: 78.30 time: 4.04
[8,   300] loss: 0.568 acc: 78.48 time: 3.84
TESTING:
Accuracy of the network on the 10000 test images: 77.48 %
Average loss on the 10000 test images: 0.625
[9,   100] loss: 0.562 acc: 77.97 time: 4.15
```

```
[9,   200] loss: 0.567 acc: 77.95 time: 3.80
[9,   300] loss: 0.562 acc: 78.56 time: 3.94
TESTING:
Accuracy of the network on the 10000 test images: 77.41 %
Average loss on the 10000 test images: 0.586
[10,   100] loss: 0.551 acc: 78.62 time: 4.00
[10,   200] loss: 0.558 acc: 77.95 time: 3.95
[10,   300] loss: 0.564 acc: 77.57 time: 4.00
TESTING:
Accuracy of the network on the 10000 test images: 77.23 %
Average loss on the 10000 test images: 0.582
[11,   100] loss: 0.554 acc: 78.39 time: 4.00
[11,   200] loss: 0.541 acc: 78.79 time: 3.98
[11,   300] loss: 0.558 acc: 78.18 time: 3.85
TESTING:
Accuracy of the network on the 10000 test images: 77.04 %
Average loss on the 10000 test images: 0.580
[12,   100] loss: 0.562 acc: 77.85 time: 4.01
[12,   200] loss: 0.548 acc: 78.71 time: 3.78
[12,   300] loss: 0.546 acc: 78.71 time: 4.17
TESTING:
Accuracy of the network on the 10000 test images: 77.54 %
Average loss on the 10000 test images: 0.572
[13,   100] loss: 0.553 acc: 78.41 time: 4.23
[13,   200] loss: 0.553 acc: 78.52 time: 3.79
[13,   300] loss: 0.547 acc: 78.74 time: 3.94
TESTING:
Accuracy of the network on the 10000 test images: 77.55 %
Average loss on the 10000 test images: 0.587
[14,   100] loss: 0.563 acc: 78.18 time: 4.04
[14,   200] loss: 0.546 acc: 78.70 time: 4.09
[14,   300] loss: 0.563 acc: 77.92 time: 3.91
TESTING:
Accuracy of the network on the 10000 test images: 77.23 %
Average loss on the 10000 test images: 0.574
[15,   100] loss: 0.565 acc: 77.73 time: 4.09
[15,   200] loss: 0.550 acc: 78.34 time: 3.94
[15,   300] loss: 0.543 acc: 78.79 time: 3.87
TESTING:
Accuracy of the network on the 10000 test images: 76.77 %
Average loss on the 10000 test images: 0.582
[16,   100] loss: 0.555 acc: 77.96 time: 4.24
[16,   200] loss: 0.557 acc: 78.41 time: 3.86
[16,   300] loss: 0.551 acc: 78.58 time: 3.93
TESTING:
Accuracy of the network on the 10000 test images: 77.21 %
Average loss on the 10000 test images: 0.581
[17,   100] loss: 0.550 acc: 78.62 time: 4.02
```

```
[17,   200] loss: 0.555 acc: 78.68 time: 3.93
[17,   300] loss: 0.558 acc: 78.27 time: 3.85
TESTING:
Accuracy of the network on the 10000 test images: 77.07 %
Average loss on the 10000 test images: 0.575
[18,   100] loss: 0.550 acc: 78.95 time: 4.19
[18,   200] loss: 0.550 acc: 78.47 time: 4.03
[18,   300] loss: 0.556 acc: 78.52 time: 3.90
TESTING:
Accuracy of the network on the 10000 test images: 77.68 %
Average loss on the 10000 test images: 0.581
[19,   100] loss: 0.558 acc: 78.30 time: 4.12
[19,   200] loss: 0.557 acc: 78.20 time: 3.92
[19,   300] loss: 0.552 acc: 78.54 time: 3.92
TESTING:
Accuracy of the network on the 10000 test images: 77.31 %
Average loss on the 10000 test images: 0.577
[20,   100] loss: 0.560 acc: 77.86 time: 4.05
[20,   200] loss: 0.549 acc: 78.82 time: 3.87
[20,   300] loss: 0.557 acc: 78.43 time: 3.83
TESTING:
Accuracy of the network on the 10000 test images: 77.53 %
Average loss on the 10000 test images: 0.580
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.

[ ]:
```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(num_classes=10).to(device)
```

[ ]:
```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
```

[ ]:
```
# 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
```

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

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

```
[1,   100] loss: 1.497 acc: 34.27 time: 4.05
[1,   200] loss: 1.320 acc: 39.25 time: 3.98
[1,   300] loss: 1.292 acc: 40.82 time: 4.03
TESTING:
Accuracy of the network on the 10000 test images: 42.77 %
Average loss on the 10000 test images: 1.248
[2,   100] loss: 1.251 acc: 43.05 time: 3.99
[2,   200] loss: 1.241 acc: 43.10 time: 3.82
```

```
[2,   300] loss: 1.236 acc: 44.05 time: 3.85
TESTING:
Accuracy of the network on the 10000 test images: 45.29 %
Average loss on the 10000 test images: 1.210
[3,   100] loss: 1.229 acc: 43.89 time: 4.06
[3,   200] loss: 1.213 acc: 45.42 time: 4.17
[3,   300] loss: 1.216 acc: 45.23 time: 3.85
TESTING:
Accuracy of the network on the 10000 test images: 48.16 %
Average loss on the 10000 test images: 1.175
[4,   100] loss: 1.202 acc: 45.69 time: 4.00
[4,   200] loss: 1.209 acc: 45.98 time: 3.89
[4,   300] loss: 1.201 acc: 45.91 time: 3.76
TESTING:
Accuracy of the network on the 10000 test images: 47.87 %
Average loss on the 10000 test images: 1.174
[5,   100] loss: 1.186 acc: 46.88 time: 4.36
[5,   200] loss: 1.192 acc: 46.15 time: 3.85
[5,   300] loss: 1.198 acc: 46.34 time: 3.87
TESTING:
Accuracy of the network on the 10000 test images: 47.59 %
Average loss on the 10000 test images: 1.169
[6,   100] loss: 1.188 acc: 47.42 time: 4.13
[6,   200] loss: 1.190 acc: 47.03 time: 3.83
[6,   300] loss: 1.184 acc: 47.32 time: 3.88
TESTING:
Accuracy of the network on the 10000 test images: 48.14 %
Average loss on the 10000 test images: 1.175
[7,   100] loss: 1.181 acc: 47.38 time: 4.04
[7,   200] loss: 1.186 acc: 46.60 time: 3.82
[7,   300] loss: 1.187 acc: 47.55 time: 3.81
TESTING:
Accuracy of the network on the 10000 test images: 48.51 %
Average loss on the 10000 test images: 1.169
[8,   100] loss: 1.181 acc: 47.48 time: 4.06
[8,   200] loss: 1.177 acc: 47.44 time: 3.89
[8,   300] loss: 1.172 acc: 48.07 time: 4.03
TESTING:
Accuracy of the network on the 10000 test images: 48.09 %
Average loss on the 10000 test images: 1.161
[9,   100] loss: 1.171 acc: 47.77 time: 4.10
[9,   200] loss: 1.169 acc: 47.73 time: 4.05
[9,   300] loss: 1.176 acc: 47.85 time: 3.93
TESTING:
Accuracy of the network on the 10000 test images: 48.54 %
Average loss on the 10000 test images: 1.162
[10,   100] loss: 1.165 acc: 48.59 time: 4.06
[10,   200] loss: 1.178 acc: 47.39 time: 4.03
```

```
[10,   300] loss: 1.175 acc: 47.45 time: 4.22
TESTING:
Accuracy of the network on the 10000 test images: 50.02 %
Average loss on the 10000 test images: 1.139
[11,   100] loss: 1.155 acc: 48.62 time: 4.12
[11,   200] loss: 1.161 acc: 48.19 time: 3.86
[11,   300] loss: 1.156 acc: 48.61 time: 3.88
TESTING:
Accuracy of the network on the 10000 test images: 50.21 %
Average loss on the 10000 test images: 1.128
[12,   100] loss: 1.146 acc: 49.23 time: 4.08
[12,   200] loss: 1.142 acc: 49.41 time: 4.18
[12,   300] loss: 1.142 acc: 49.21 time: 4.02
TESTING:
Accuracy of the network on the 10000 test images: 50.10 %
Average loss on the 10000 test images: 1.129
[13,   100] loss: 1.128 acc: 50.70 time: 4.00
[13,   200] loss: 1.138 acc: 49.66 time: 3.94
[13,   300] loss: 1.129 acc: 50.97 time: 3.76
TESTING:
Accuracy of the network on the 10000 test images: 51.12 %
Average loss on the 10000 test images: 1.120
[14,   100] loss: 1.131 acc: 50.15 time: 4.21
[14,   200] loss: 1.136 acc: 50.14 time: 3.77
[14,   300] loss: 1.128 acc: 50.20 time: 3.94
TESTING:
Accuracy of the network on the 10000 test images: 50.64 %
Average loss on the 10000 test images: 1.115
[15,   100] loss: 1.128 acc: 50.58 time: 3.93
[15,   200] loss: 1.132 acc: 50.04 time: 3.80
[15,   300] loss: 1.129 acc: 50.13 time: 3.77
TESTING:
Accuracy of the network on the 10000 test images: 51.08 %
Average loss on the 10000 test images: 1.113
[16,   100] loss: 1.130 acc: 50.41 time: 3.91
[16,   200] loss: 1.131 acc: 49.98 time: 3.87
[16,   300] loss: 1.142 acc: 49.53 time: 3.81
TESTING:
Accuracy of the network on the 10000 test images: 51.15 %
Average loss on the 10000 test images: 1.113
[17,   100] loss: 1.130 acc: 50.39 time: 3.96
[17,   200] loss: 1.136 acc: 49.74 time: 3.94
[17,   300] loss: 1.127 acc: 50.23 time: 3.90
TESTING:
Accuracy of the network on the 10000 test images: 51.76 %
Average loss on the 10000 test images: 1.103
[18,   100] loss: 1.119 acc: 51.02 time: 4.01
[18,   200] loss: 1.124 acc: 50.58 time: 3.87
```

```
[18,   300] loss: 1.122 acc: 50.82 time: 3.87
TESTING:
Accuracy of the network on the 10000 test images: 51.37 %
Average loss on the 10000 test images: 1.107
[19,   100] loss: 1.130 acc: 50.27 time: 4.19
[19,   200] loss: 1.121 acc: 50.67 time: 3.88
[19,   300] loss: 1.125 acc: 50.92 time: 4.20
TESTING:
Accuracy of the network on the 10000 test images: 51.73 %
Average loss on the 10000 test images: 1.107
[20,   100] loss: 1.129 acc: 49.91 time: 4.02
[20,   200] loss: 1.123 acc: 50.71 time: 3.91
[20,   300] loss: 1.119 acc: 50.91 time: 3.91
TESTING:
Accuracy of the network on the 10000 test images: 51.52 %
Average loss on the 10000 test images: 1.106
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.

[ ]:
```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(num_classes=10).to(device)
net_params = torch.load(PATH)
net_params['fc.weight'] = torch.randn(10, net_params['fc.weight'].shape[1])
net_params['fc.bias'] = torch.randn(10)
net.load_state_dict(net_params)
```

[ ]:
```
<All keys matched successfully>
```

[ ]:
```python
# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

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

```
[1,   100] loss: 1.027 acc: 66.83 time: 4.87
[1,   200] loss: 0.687 acc: 73.51 time: 4.50
[1,   300] loss: 0.696 acc: 73.00 time: 4.41
TESTING:
```

```
Accuracy of the network on the 10000 test images: 69.39 %
Average loss on the 10000 test images: 0.777
[2,   100] loss: 1.086 acc: 59.95 time: 4.75
[2,   200] loss: 1.160 acc: 54.30 time: 4.58
[2,   300] loss: 1.000 acc: 59.91 time: 4.49
TESTING:
Accuracy of the network on the 10000 test images: 67.05 %
Average loss on the 10000 test images: 0.819
[3,   100] loss: 0.794 acc: 68.13 time: 4.77
[3,   200] loss: 0.738 acc: 70.16 time: 4.54
[3,   300] loss: 0.817 acc: 68.14 time: 4.49
TESTING:
Accuracy of the network on the 10000 test images: 69.73 %
Average loss on the 10000 test images: 0.768
[4,   100] loss: 0.798 acc: 68.77 time: 4.62
[4,   200] loss: 0.736 acc: 70.93 time: 4.56
[4,   300] loss: 0.693 acc: 72.46 time: 4.45
TESTING:
Accuracy of the network on the 10000 test images: 72.89 %
Average loss on the 10000 test images: 0.711
[5,   100] loss: 0.673 acc: 73.85 time: 4.69
[5,   200] loss: 0.662 acc: 73.56 time: 4.57
[5,   300] loss: 0.881 acc: 65.99 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 65.98 %
Average loss on the 10000 test images: 0.871
[6,   100] loss: 0.831 acc: 67.58 time: 4.72
[6,   200] loss: 0.762 acc: 69.71 time: 4.52
[6,   300] loss: 0.714 acc: 71.62 time: 4.51
TESTING:
Accuracy of the network on the 10000 test images: 73.52 %
Average loss on the 10000 test images: 0.674
[7,   100] loss: 0.670 acc: 73.58 time: 4.64
[7,   200] loss: 0.654 acc: 74.30 time: 4.47
[7,   300] loss: 0.653 acc: 74.23 time: 4.59
TESTING:
Accuracy of the network on the 10000 test images: 74.44 %
Average loss on the 10000 test images: 0.648
[8,   100] loss: 0.644 acc: 74.74 time: 4.66
[8,   200] loss: 0.644 acc: 74.44 time: 4.68
[8,   300] loss: 0.632 acc: 74.94 time: 4.58
TESTING:
Accuracy of the network on the 10000 test images: 74.80 %
Average loss on the 10000 test images: 0.634
[9,   100] loss: 0.622 acc: 75.57 time: 4.80
[9,   200] loss: 0.639 acc: 74.74 time: 4.50
[9,   300] loss: 0.638 acc: 74.97 time: 4.51
TESTING:
```

```
Accuracy of the network on the 10000 test images: 75.71 %
Average loss on the 10000 test images: 0.616
[10,   100] loss: 0.626 acc: 75.38 time: 4.67
[10,   200] loss: 0.626 acc: 75.24 time: 4.47
[10,   300] loss: 0.677 acc: 73.94 time: 4.49
TESTING:
Accuracy of the network on the 10000 test images: 74.46 %
Average loss on the 10000 test images: 0.665
[11,   100] loss: 0.635 acc: 75.28 time: 4.67
[11,   200] loss: 0.625 acc: 75.40 time: 4.54
[11,   300] loss: 0.611 acc: 76.08 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 76.14 %
Average loss on the 10000 test images: 0.611
[12,   100] loss: 0.583 acc: 77.27 time: 4.70
[12,   200] loss: 0.589 acc: 77.06 time: 4.57
[12,   300] loss: 0.582 acc: 77.59 time: 4.45
TESTING:
Accuracy of the network on the 10000 test images: 76.69 %
Average loss on the 10000 test images: 0.587
[13,   100] loss: 0.567 acc: 78.01 time: 4.74
[13,   200] loss: 0.578 acc: 77.38 time: 4.48
[13,   300] loss: 0.586 acc: 77.18 time: 4.46
TESTING:
Accuracy of the network on the 10000 test images: 77.19 %
Average loss on the 10000 test images: 0.586
[14,   100] loss: 0.567 acc: 77.84 time: 4.71
[14,   200] loss: 0.572 acc: 77.65 time: 4.48
[14,   300] loss: 0.559 acc: 78.34 time: 4.54
TESTING:
Accuracy of the network on the 10000 test images: 77.89 %
Average loss on the 10000 test images: 0.580
[15,   100] loss: 0.560 acc: 78.16 time: 4.69
[15,   200] loss: 0.557 acc: 78.35 time: 4.54
[15,   300] loss: 0.545 acc: 78.88 time: 4.55
TESTING:
Accuracy of the network on the 10000 test images: 77.58 %
Average loss on the 10000 test images: 0.576
[16,   100] loss: 0.545 acc: 78.75 time: 4.77
[16,   200] loss: 0.552 acc: 78.52 time: 4.48
[16,   300] loss: 0.548 acc: 78.29 time: 4.48
TESTING:
Accuracy of the network on the 10000 test images: 78.00 %
Average loss on the 10000 test images: 0.565
[17,   100] loss: 0.550 acc: 78.37 time: 4.73
[17,   200] loss: 0.541 acc: 79.02 time: 4.50
[17,   300] loss: 0.539 acc: 78.99 time: 4.65
TESTING:
```

```
Accuracy of the network on the 10000 test images: 77.58 %
Average loss on the 10000 test images: 0.566
[18,   100] loss: 0.531 acc: 79.48 time: 4.70
[18,   200] loss: 0.550 acc: 78.56 time: 4.53
[18,   300] loss: 0.531 acc: 79.94 time: 4.59
TESTING:
Accuracy of the network on the 10000 test images: 78.16 %
Average loss on the 10000 test images: 0.556
[19,   100] loss: 0.533 acc: 78.99 time: 4.70
[19,   200] loss: 0.527 acc: 79.09 time: 4.49
[19,   300] loss: 0.535 acc: 79.15 time: 4.51
TESTING:
Accuracy of the network on the 10000 test images: 78.19 %
Average loss on the 10000 test images: 0.553
[20,   100] loss: 0.545 acc: 78.84 time: 4.72
[20,   200] loss: 0.523 acc: 79.82 time: 4.57
[20,   300] loss: 0.528 acc: 79.77 time: 4.51
TESTING:
Accuracy of the network on the 10000 test images: 79.08 %
Average loss on the 10000 test images: 0.542
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.

[ ]:
```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(num_classes=10, pretrained=False).to(device)
```

```
/opt/conda/lib/python3.10/site-packages/torchvision/models/_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=None`.
  warnings.warn(msg)
```

[ ]:
```python
# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

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

```
[1,   100] loss: 1.508 acc: 34.16 time: 4.76
[1,   200] loss: 1.253 acc: 44.12 time: 4.51
[1,   300] loss: 1.170 acc: 48.21 time: 4.45
TESTING:
Accuracy of the network on the 10000 test images: 53.33 %
Average loss on the 10000 test images: 1.080
[2,   100] loss: 1.116 acc: 51.80 time: 4.71
[2,   200] loss: 1.086 acc: 52.89 time: 4.54
[2,   300] loss: 1.073 acc: 53.77 time: 4.47
TESTING:
Accuracy of the network on the 10000 test images: 55.22 %
Average loss on the 10000 test images: 1.048
[3,   100] loss: 1.050 acc: 55.53 time: 4.78
[3,   200] loss: 1.026 acc: 56.73 time: 4.72
[3,   300] loss: 1.018 acc: 56.48 time: 4.54
TESTING:
Accuracy of the network on the 10000 test images: 57.64 %
Average loss on the 10000 test images: 0.983
[4,   100] loss: 1.001 acc: 57.87 time: 4.64
[4,   200] loss: 0.975 acc: 58.57 time: 4.51
[4,   300] loss: 0.972 acc: 59.16 time: 4.51
TESTING:
Accuracy of the network on the 10000 test images: 61.57 %
Average loss on the 10000 test images: 0.928
[5,   100] loss: 0.944 acc: 60.71 time: 4.71
[5,   200] loss: 0.926 acc: 61.02 time: 4.59
[5,   300] loss: 0.935 acc: 61.09 time: 4.54
TESTING:
Accuracy of the network on the 10000 test images: 62.84 %
Average loss on the 10000 test images: 0.880
[6,   100] loss: 0.916 acc: 62.19 time: 4.77
[6,   200] loss: 0.903 acc: 62.12 time: 4.54
[6,   300] loss: 0.908 acc: 61.98 time: 4.61
TESTING:
Accuracy of the network on the 10000 test images: 63.62 %
Average loss on the 10000 test images: 0.881
[7,   100] loss: 0.875 acc: 63.51 time: 4.69
[7,   200] loss: 0.879 acc: 63.23 time: 4.48
[7,   300] loss: 0.891 acc: 63.52 time: 4.57
TESTING:
Accuracy of the network on the 10000 test images: 65.10 %
Average loss on the 10000 test images: 0.846
```

```
[8,   100] loss: 0.861 acc: 64.14 time: 4.78
[8,   200] loss: 0.857 acc: 65.20 time: 4.61
[8,   300] loss: 0.843 acc: 65.22 time: 4.46
TESTING:
Accuracy of the network on the 10000 test images: 66.46 %
Average loss on the 10000 test images: 0.821
[9,   100] loss: 0.833 acc: 65.98 time: 4.68
[9,   200] loss: 0.844 acc: 65.23 time: 4.51
[9,   300] loss: 0.827 acc: 66.41 time: 4.47
TESTING:
Accuracy of the network on the 10000 test images: 67.32 %
Average loss on the 10000 test images: 0.791
[10,   100] loss: 0.826 acc: 65.77 time: 4.70
[10,   200] loss: 0.809 acc: 66.89 time: 4.48
[10,   300] loss: 0.813 acc: 66.72 time: 4.49
TESTING:
Accuracy of the network on the 10000 test images: 67.52 %
Average loss on the 10000 test images: 0.802
[11,   100] loss: 0.773 acc: 69.10 time: 4.66
[11,   200] loss: 0.739 acc: 70.09 time: 4.62
[11,   300] loss: 0.734 acc: 70.84 time: 4.48
TESTING:
Accuracy of the network on the 10000 test images: 70.81 %
Average loss on the 10000 test images: 0.725
[12,   100] loss: 0.728 acc: 70.77 time: 4.63
[12,   200] loss: 0.721 acc: 70.57 time: 4.51
[12,   300] loss: 0.718 acc: 71.24 time: 4.47
TESTING:
Accuracy of the network on the 10000 test images: 70.75 %
Average loss on the 10000 test images: 0.719
[13,   100] loss: 0.707 acc: 71.74 time: 4.68
[13,   200] loss: 0.717 acc: 71.48 time: 4.51
[13,   300] loss: 0.698 acc: 71.83 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 71.97 %
Average loss on the 10000 test images: 0.709
[14,   100] loss: 0.695 acc: 71.88 time: 4.77
[14,   200] loss: 0.697 acc: 71.79 time: 4.74
[14,   300] loss: 0.694 acc: 72.23 time: 4.52
TESTING:
Accuracy of the network on the 10000 test images: 71.72 %
Average loss on the 10000 test images: 0.696
[15,   100] loss: 0.699 acc: 72.27 time: 4.71
[15,   200] loss: 0.691 acc: 72.27 time: 4.54
[15,   300] loss: 0.688 acc: 72.13 time: 4.64
TESTING:
Accuracy of the network on the 10000 test images: 72.35 %
Average loss on the 10000 test images: 0.690
```

```
[16,   100] loss: 0.694 acc: 72.17 time: 4.73
[16,   200] loss: 0.684 acc: 72.34 time: 4.48
[16,   300] loss: 0.677 acc: 73.16 time: 4.47
TESTING:
Accuracy of the network on the 10000 test images: 72.41 %
Average loss on the 10000 test images: 0.688
[17,   100] loss: 0.668 acc: 73.29 time: 4.64
[17,   200] loss: 0.686 acc: 72.54 time: 4.57
[17,   300] loss: 0.666 acc: 73.66 time: 4.59
TESTING:
Accuracy of the network on the 10000 test images: 72.76 %
Average loss on the 10000 test images: 0.680
[18,   100] loss: 0.668 acc: 73.50 time: 4.69
[18,   200] loss: 0.662 acc: 73.17 time: 4.51
[18,   300] loss: 0.681 acc: 72.62 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 73.62 %
Average loss on the 10000 test images: 0.662
[19,   100] loss: 0.668 acc: 73.22 time: 4.78
[19,   200] loss: 0.655 acc: 73.96 time: 4.49
[19,   300] loss: 0.678 acc: 73.14 time: 4.49
TESTING:
Accuracy of the network on the 10000 test images: 73.27 %
Average loss on the 10000 test images: 0.661
[20,   100] loss: 0.664 acc: 73.18 time: 4.76
[20,   200] loss: 0.647 acc: 74.41 time: 4.55
[20,   300] loss: 0.655 acc: 74.14 time: 4.52
TESTING:
Accuracy of the network on the 10000 test images: 73.70 %
Average loss on the 10000 test images: 0.657
Finished Training
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[24]:
```
!jupyter nbconvert --to pdf /content/drive/MyDrive/CS444/CS444HW3/
 ↪a3_part1_rotation.ipynb
```

```
[NbConvertApp] WARNING | pattern
'/content/drive/MyDrive/CS444/CS444HW3/a3_part1_rotation.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.
```

```
Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
```

```
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
```

```
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook.
To recover
                                    previous default behaviour (outputting to the
current
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
```

```
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
```

```
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```