a3_part1_rotation (3)

April 6, 2023

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_starter/assignment3_part1"
if not os.path.exists(datadir):
  !ln -s "/content/drive/MyDrive/assignment3_starter/assignment3_part1"␣
↪$datadir # TODO: Fill your A3 path
os.chdir(datadir)
!pwd
#"/content/drive/My Drive/path/to/your/assignment3_starter/assignment3_part1"
#/content/drive/MyDrive/assignment3_starter/assignment3_part1
```

```
/content/drive/MyDrive/assignment3_starter/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 `__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
    elif rot == 1: # 90 degrees rotation
        return torch.rot90(img.permute(1,2,0), 3).permute(2,0,1)
    elif rot == 2: # 180 degrees rotation
        return torch.rot90(img, 2)
    elif rot == 3: # 270 degrees rotation
        return torch.rot90(img.permute(1,2,0), 1).permute(2,0,1)

    # TODO: Implement rotate_img() - return the rotated img
    #
    #
    #
    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)
```

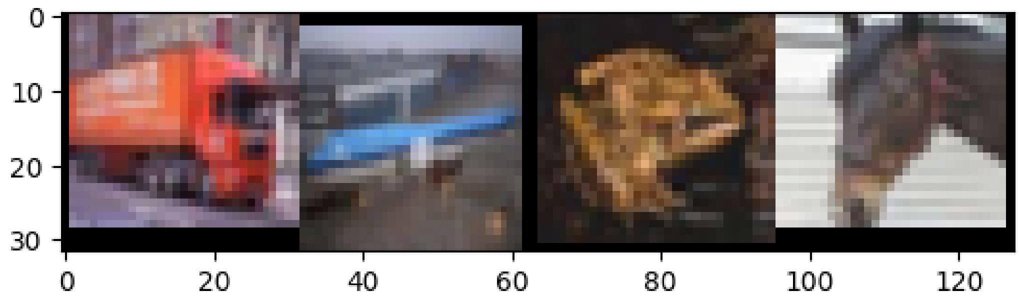
```
Files already downloaded and verified
Files already downloaded and verified
```

Show some example images and rotated images with labels:

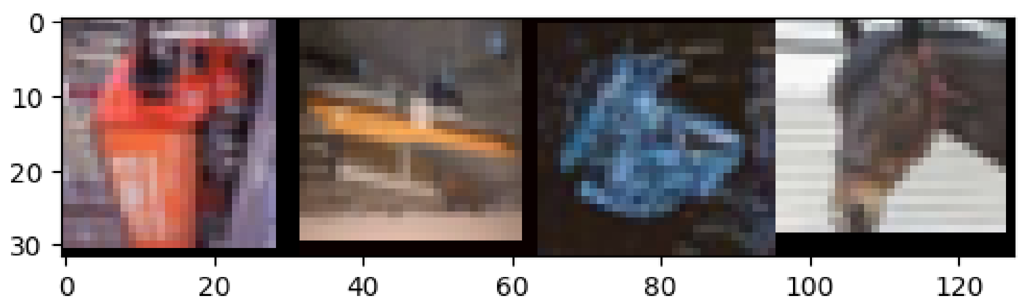
[ ]:
```
import matplotlib.pyplot as plt

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)))
```

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




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

Class labels:  truck plane frog  horse
```




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

#Evaluation code

[ ]:
```
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
            #
            #
            #
            # Calculate outputs by running images through the network


            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)

            #correct += labels.size(0)
            total += labels.size(0)
            # loss
            correct += (predicted == labels).sum().item()
            avg_test_loss += criterion(outputs, labels)  / len(testloader)

            # doubt on below line
            #total += avg_test_loss


    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}')
```

[ ]:
```
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
```

#Train a ResNet18 on the rotation task

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' 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
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
```

[ ]:
```
# 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()
        adjust_learning_rate(optimizer, epoch, init_lr, decay_epochs)


        for i, (imgs, imgs_rotated, rotation_label, cls_label) in␣
↪enumerate(trainloader, 0):


            # TODO: Set the data to the correct device; Different task will use␣
↪different inputs and labels
            #

            # TODO: Zero the parameter gradients
            #

            # TODO: forward + backward + optimize
            #
            #
            #
            #
```

```
            #
            # Set the data to the correct device; Different task will use␣
↪different inputs and labels
            if task == 'rotation':
              images, labels = imgs_rotated.to(device), rotation_label.
↪to(device)
            elif task == 'classification':
              images, labels = imgs.to(device), cls_label.to(device)

            # Zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            # TODO: Get predicted results

            predicted = torch.argmax(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')
```

[ ]:
```
train(net, criterion, optimizer, num_epochs=45, decay_epochs=15, init_lr=0.1,
    task='rotation')

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

```
[1,   100] loss: 0.630 acc: 71.41 time: 7.69
[1,   200] loss: 0.627 acc: 72.17 time: 5.96
[1,   300] loss: 0.619 acc: 72.41 time: 7.33
TESTING:
Accuracy of the network on the 10000 test images: 64.59 %
Average loss on the 10000 test images: 0.937
[2,   100] loss: 0.602 acc: 73.03 time: 7.58
[2,   200] loss: 0.611 acc: 72.48 time: 5.75
[2,   300] loss: 0.608 acc: 72.95 time: 7.37
TESTING:
Accuracy of the network on the 10000 test images: 63.24 %
Average loss on the 10000 test images: 0.902
[3,   100] loss: 0.587 acc: 74.07 time: 7.20
[3,   200] loss: 0.586 acc: 74.17 time: 5.70
[3,   300] loss: 0.588 acc: 73.80 time: 7.36
TESTING:
Accuracy of the network on the 10000 test images: 67.88 %
Average loss on the 10000 test images: 0.808
[4,   100] loss: 0.587 acc: 74.11 time: 6.75
[4,   200] loss: 0.591 acc: 73.95 time: 6.10
[4,   300] loss: 0.584 acc: 74.16 time: 7.07
TESTING:
Accuracy of the network on the 10000 test images: 62.32 %
Average loss on the 10000 test images: 0.953
[5,   100] loss: 0.572 acc: 74.82 time: 6.06
[5,   200] loss: 0.575 acc: 74.60 time: 6.96
[5,   300] loss: 0.565 acc: 75.31 time: 6.26
TESTING:
Accuracy of the network on the 10000 test images: 65.57 %
Average loss on the 10000 test images: 0.835
[6,   100] loss: 0.570 acc: 75.00 time: 8.03
[6,   200] loss: 0.568 acc: 74.92 time: 7.57
[6,   300] loss: 0.561 acc: 74.83 time: 5.67
TESTING:
Accuracy of the network on the 10000 test images: 64.82 %
Average loss on the 10000 test images: 0.883
[7,   100] loss: 0.553 acc: 75.59 time: 6.02
[7,   200] loss: 0.564 acc: 75.16 time: 7.50
[7,   300] loss: 0.555 acc: 75.56 time: 5.72
TESTING:
```

```
Accuracy of the network on the 10000 test images: 67.59 %
Average loss on the 10000 test images: 0.797
[8,   100] loss: 0.557 acc: 75.73 time: 7.01
[8,   200] loss: 0.544 acc: 76.05 time: 6.34
[8,   300] loss: 0.547 acc: 76.25 time: 6.05
TESTING:
Accuracy of the network on the 10000 test images: 65.89 %
Average loss on the 10000 test images: 0.837
[9,   100] loss: 0.536 acc: 76.88 time: 7.55
[9,   200] loss: 0.565 acc: 75.25 time: 5.82
[9,   300] loss: 0.544 acc: 76.52 time: 6.85
TESTING:
Accuracy of the network on the 10000 test images: 66.15 %
Average loss on the 10000 test images: 0.833
[10,   100] loss: 0.537 acc: 76.48 time: 7.61
[10,   200] loss: 0.557 acc: 75.48 time: 5.88
[10,   300] loss: 0.553 acc: 76.03 time: 7.29
TESTING:
Accuracy of the network on the 10000 test images: 67.56 %
Average loss on the 10000 test images: 0.817
[11,   100] loss: 0.534 acc: 76.70 time: 7.65
[11,   200] loss: 0.555 acc: 75.98 time: 5.92
[11,   300] loss: 0.547 acc: 76.11 time: 7.61
TESTING:
Accuracy of the network on the 10000 test images: 66.04 %
Average loss on the 10000 test images: 0.846
[12,   100] loss: 0.543 acc: 76.47 time: 7.26
[12,   200] loss: 0.539 acc: 77.02 time: 5.84
[12,   300] loss: 0.538 acc: 77.01 time: 7.29
TESTING:
Accuracy of the network on the 10000 test images: 66.18 %
Average loss on the 10000 test images: 0.860
[13,   100] loss: 0.535 acc: 76.68 time: 6.22
[13,   200] loss: 0.537 acc: 76.37 time: 6.54
[13,   300] loss: 0.537 acc: 76.73 time: 6.71
TESTING:
Accuracy of the network on the 10000 test images: 65.51 %
Average loss on the 10000 test images: 0.921
[14,   100] loss: 0.527 acc: 76.86 time: 6.18
[14,   200] loss: 0.543 acc: 76.90 time: 7.51
[14,   300] loss: 0.529 acc: 76.96 time: 5.82
TESTING:
Accuracy of the network on the 10000 test images: 68.38 %
Average loss on the 10000 test images: 0.812
[15,   100] loss: 0.535 acc: 77.02 time: 5.93
[15,   200] loss: 0.527 acc: 77.46 time: 7.44
[15,   300] loss: 0.520 acc: 77.62 time: 5.91
TESTING:
```

```
Accuracy of the network on the 10000 test images: 66.16 %
Average loss on the 10000 test images: 0.836
[16,   100] loss: 0.479 acc: 79.52 time: 6.12
[16,   200] loss: 0.448 acc: 80.77 time: 7.40
[16,   300] loss: 0.430 acc: 81.89 time: 5.73
TESTING:
Accuracy of the network on the 10000 test images: 73.64 %
Average loss on the 10000 test images: 0.646
[17,   100] loss: 0.425 acc: 81.94 time: 6.74
[17,   200] loss: 0.424 acc: 82.05 time: 6.77
[17,   300] loss: 0.420 acc: 82.10 time: 6.12
TESTING:
Accuracy of the network on the 10000 test images: 74.57 %
Average loss on the 10000 test images: 0.640
[18,   100] loss: 0.418 acc: 82.05 time: 7.68
[18,   200] loss: 0.410 acc: 83.15 time: 5.87
[18,   300] loss: 0.405 acc: 83.08 time: 6.68
TESTING:
Accuracy of the network on the 10000 test images: 76.77 %
Average loss on the 10000 test images: 0.587
[19,   100] loss: 0.410 acc: 82.80 time: 7.71
[19,   200] loss: 0.405 acc: 82.77 time: 5.84
[19,   300] loss: 0.399 acc: 83.27 time: 7.44
TESTING:
Accuracy of the network on the 10000 test images: 76.03 %
Average loss on the 10000 test images: 0.602
[20,   100] loss: 0.401 acc: 83.30 time: 7.69
[20,   200] loss: 0.407 acc: 82.91 time: 5.69
[20,   300] loss: 0.386 acc: 84.20 time: 7.42
TESTING:
Accuracy of the network on the 10000 test images: 76.79 %
Average loss on the 10000 test images: 0.582
[21,   100] loss: 0.399 acc: 83.24 time: 7.59
[21,   200] loss: 0.394 acc: 83.49 time: 5.83
[21,   300] loss: 0.384 acc: 84.17 time: 7.50
TESTING:
Accuracy of the network on the 10000 test images: 76.30 %
Average loss on the 10000 test images: 0.598
[22,   100] loss: 0.380 acc: 84.08 time: 6.92
[22,   200] loss: 0.386 acc: 83.87 time: 5.96
[22,   300] loss: 0.393 acc: 83.57 time: 7.37
TESTING:
Accuracy of the network on the 10000 test images: 77.03 %
Average loss on the 10000 test images: 0.577
[23,   100] loss: 0.375 acc: 84.53 time: 6.18
[23,   200] loss: 0.383 acc: 83.84 time: 6.65
[23,   300] loss: 0.372 acc: 84.60 time: 6.69
TESTING:
```

```
Accuracy of the network on the 10000 test images: 77.62 %
Average loss on the 10000 test images: 0.570
[24,   100] loss: 0.374 acc: 84.66 time: 5.99
[24,   200] loss: 0.379 acc: 84.38 time: 7.38
[24,   300] loss: 0.373 acc: 84.62 time: 5.90
TESTING:
Accuracy of the network on the 10000 test images: 77.93 %
Average loss on the 10000 test images: 0.557
[25,   100] loss: 0.374 acc: 84.50 time: 5.82
[25,   200] loss: 0.368 acc: 84.77 time: 7.37
[25,   300] loss: 0.377 acc: 84.02 time: 5.75
TESTING:
Accuracy of the network on the 10000 test images: 77.67 %
Average loss on the 10000 test images: 0.566
[26,   100] loss: 0.376 acc: 84.23 time: 5.84
[26,   200] loss: 0.365 acc: 84.85 time: 7.27
[26,   300] loss: 0.371 acc: 84.90 time: 5.98
TESTING:
Accuracy of the network on the 10000 test images: 77.73 %
Average loss on the 10000 test images: 0.562
[27,   100] loss: 0.368 acc: 84.98 time: 5.87
[27,   200] loss: 0.356 acc: 85.25 time: 7.27
[27,   300] loss: 0.366 acc: 85.01 time: 5.89
TESTING:
Accuracy of the network on the 10000 test images: 76.45 %
Average loss on the 10000 test images: 0.598
[28,   100] loss: 0.355 acc: 85.33 time: 6.18
[28,   200] loss: 0.357 acc: 85.21 time: 7.16
[28,   300] loss: 0.361 acc: 85.30 time: 5.81
TESTING:
Accuracy of the network on the 10000 test images: 78.63 %
Average loss on the 10000 test images: 0.558
[29,   100] loss: 0.360 acc: 85.42 time: 6.80
[29,   200] loss: 0.351 acc: 85.50 time: 6.53
[29,   300] loss: 0.362 acc: 84.83 time: 5.95
TESTING:
Accuracy of the network on the 10000 test images: 78.40 %
Average loss on the 10000 test images: 0.558
[30,   100] loss: 0.353 acc: 85.36 time: 7.49
[30,   200] loss: 0.358 acc: 85.46 time: 5.88
[30,   300] loss: 0.366 acc: 84.78 time: 7.03
TESTING:
Accuracy of the network on the 10000 test images: 77.04 %
Average loss on the 10000 test images: 0.594
[31,   100] loss: 0.349 acc: 85.58 time: 7.64
[31,   200] loss: 0.345 acc: 85.62 time: 5.67
[31,   300] loss: 0.332 acc: 86.60 time: 7.52
TESTING:
```

```
Accuracy of the network on the 10000 test images: 79.48 %
Average loss on the 10000 test images: 0.526
[32,   100] loss: 0.326 acc: 86.55 time: 7.57
[32,   200] loss: 0.314 acc: 87.27 time: 5.68
[32,   300] loss: 0.323 acc: 87.00 time: 7.40
TESTING:
Accuracy of the network on the 10000 test images: 79.95 %
Average loss on the 10000 test images: 0.514
[33,   100] loss: 0.318 acc: 86.90 time: 7.40
[33,   200] loss: 0.322 acc: 86.68 time: 5.77
[33,   300] loss: 0.320 acc: 86.84 time: 7.38
TESTING:
Accuracy of the network on the 10000 test images: 79.86 %
Average loss on the 10000 test images: 0.518
[34,   100] loss: 0.324 acc: 86.83 time: 6.50
[34,   200] loss: 0.323 acc: 86.98 time: 6.45
[34,   300] loss: 0.321 acc: 86.75 time: 6.89
TESTING:
Accuracy of the network on the 10000 test images: 80.95 %
Average loss on the 10000 test images: 0.497
[35,   100] loss: 0.314 acc: 87.20 time: 5.90
[35,   200] loss: 0.322 acc: 86.79 time: 7.19
[35,   300] loss: 0.312 acc: 87.29 time: 5.95
TESTING:
Accuracy of the network on the 10000 test images: 80.80 %
Average loss on the 10000 test images: 0.495
[36,   100] loss: 0.303 acc: 87.92 time: 5.97
[36,   200] loss: 0.305 acc: 87.50 time: 7.31
[36,   300] loss: 0.319 acc: 87.16 time: 5.82
TESTING:
Accuracy of the network on the 10000 test images: 80.61 %
Average loss on the 10000 test images: 0.498
[37,   100] loss: 0.312 acc: 87.52 time: 6.19
[37,   200] loss: 0.308 acc: 87.46 time: 7.34
[37,   300] loss: 0.308 acc: 87.65 time: 5.91
[38,   100] loss: 0.311 acc: 87.54 time: 6.70
[38,   200] loss: 0.313 acc: 87.20 time: 6.79
[38,   300] loss: 0.308 acc: 87.55 time: 5.74
TESTING:
Accuracy of the network on the 10000 test images: 81.26 %
Average loss on the 10000 test images: 0.486
[39,   100] loss: 0.311 acc: 87.64 time: 7.65
[39,   200] loss: 0.303 acc: 87.62 time: 5.85
[39,   300] loss: 0.316 acc: 87.23 time: 7.01
TESTING:
Accuracy of the network on the 10000 test images: 81.45 %
Average loss on the 10000 test images: 0.489
[40,   100] loss: 0.307 acc: 87.54 time: 7.72
```

```
[40,   200] loss: 0.313 acc: 87.20 time: 5.80
[40,   300] loss: 0.308 acc: 87.73 time: 7.40
TESTING:
Accuracy of the network on the 10000 test images: 81.10 %
Average loss on the 10000 test images: 0.485
[41,   100] loss: 0.308 acc: 87.46 time: 7.73
[41,   200] loss: 0.298 acc: 87.79 time: 5.74
[41,   300] loss: 0.302 acc: 87.77 time: 7.44
TESTING:
Accuracy of the network on the 10000 test images: 81.39 %
Average loss on the 10000 test images: 0.475
[42,   100] loss: 0.303 acc: 87.72 time: 7.68
[42,   200] loss: 0.310 acc: 87.38 time: 5.84
[42,   300] loss: 0.309 acc: 87.52 time: 7.38
TESTING:
Accuracy of the network on the 10000 test images: 81.81 %
Average loss on the 10000 test images: 0.478
[43,   100] loss: 0.307 acc: 87.60 time: 6.63
[43,   200] loss: 0.305 acc: 87.60 time: 6.38
[43,   300] loss: 0.312 acc: 87.35 time: 6.69
TESTING:
Accuracy of the network on the 10000 test images: 81.84 %
Average loss on the 10000 test images: 0.469
[44,   100] loss: 0.298 acc: 87.89 time: 5.76
[44,   200] loss: 0.297 acc: 88.00 time: 6.86
[44,   300] loss: 0.302 acc: 87.76 time: 6.20
TESTING:
Accuracy of the network on the 10000 test images: 80.52 %
Average loss on the 10000 test images: 0.503
[45,   100] loss: 0.300 acc: 87.72 time: 5.98
[45,   200] loss: 0.297 acc: 87.78 time: 7.20
[45,   300] loss: 0.306 acc: 87.72 time: 5.82
TESTING:
Accuracy of the network on the 10000 test images: 81.51 %
Average loss on the 10000 test images: 0.473
Finished Training
```

##Fine-tuning on the pre-trained model

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.

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

from torchvision import models
```

```
# Load the previously trained ResNet18 model
net = models.resnet18(num_classes=4)

saved_path = 'rotation_model.pth'
net.load_state_dict(torch.load(saved_path))

# Move the model to the device (GPU or CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#net.to(device)


num_classes = 10
net.fc = nn.Linear(net.fc.in_features, num_classes).to(device)
```

[ ]:
```
# num_classes = 10
# net.fc = nn.Linear(net.fc.in_features, num_classes)
```

[ ]:
```
for param in net.parameters():
    param.requires_grad = False


# Unfreeze the 'layer4' block and 'fc' layer
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
```

[ ]:
```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(params_to_update, lr=0.01)
#num_epochs = 10
#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = net.to(device)
```

[ ]:
```python
train(net, criterion, optimizer, num_epochs=30, decay_epochs=10, init_lr=0.1,
 ↪task='classification')
```

```
[1,   100] loss: 1.948 acc: 30.16 time: 12.81
[1,   200] loss: 1.644 acc: 39.15 time: 7.16
[1,   300] loss: 1.597 acc: 41.70 time: 5.55
TESTING:
Accuracy of the network on the 10000 test images: 43.95 %
Average loss on the 10000 test images: 1.541
[2,   100] loss: 1.517 acc: 44.62 time: 5.62
[2,   200] loss: 1.510 acc: 44.86 time: 7.31
[2,   300] loss: 1.527 acc: 44.70 time: 5.43
TESTING:
Accuracy of the network on the 10000 test images: 47.27 %
Average loss on the 10000 test images: 1.478
[3,   100] loss: 1.474 acc: 46.41 time: 5.64
[3,   200] loss: 1.482 acc: 46.03 time: 7.02
[3,   300] loss: 1.454 acc: 47.38 time: 5.76
TESTING:
Accuracy of the network on the 10000 test images: 48.26 %
Average loss on the 10000 test images: 1.435
[4,   100] loss: 1.458 acc: 48.16 time: 5.72
[4,   200] loss: 1.447 acc: 47.85 time: 7.14
[4,   300] loss: 1.454 acc: 47.47 time: 5.57
TESTING:
Accuracy of the network on the 10000 test images: 48.56 %
Average loss on the 10000 test images: 1.436
[5,   100] loss: 1.413 acc: 48.76 time: 5.71
[5,   200] loss: 1.426 acc: 48.64 time: 7.31
[5,   300] loss: 1.431 acc: 48.14 time: 5.59
TESTING:
Accuracy of the network on the 10000 test images: 47.90 %
Average loss on the 10000 test images: 1.454
```

```
[6,   100] loss: 1.415 acc: 49.30 time: 5.75
[6,   200] loss: 1.416 acc: 49.84 time: 7.30
[6,   300] loss: 1.394 acc: 50.17 time: 5.66
TESTING:
Accuracy of the network on the 10000 test images: 52.09 %
Average loss on the 10000 test images: 1.342
[7,   100] loss: 1.426 acc: 49.08 time: 5.87
[7,   200] loss: 1.398 acc: 49.56 time: 7.38
[7,   300] loss: 1.404 acc: 49.54 time: 5.64
TESTING:
Accuracy of the network on the 10000 test images: 52.12 %
Average loss on the 10000 test images: 1.341
[8,   100] loss: 1.387 acc: 50.69 time: 5.79
[8,   200] loss: 1.403 acc: 49.49 time: 7.47
[8,   300] loss: 1.400 acc: 50.16 time: 5.57
TESTING:
Accuracy of the network on the 10000 test images: 53.26 %
Average loss on the 10000 test images: 1.305
[9,   100] loss: 1.391 acc: 50.50 time: 5.73
[9,   200] loss: 1.394 acc: 50.27 time: 7.30
[9,   300] loss: 1.392 acc: 50.83 time: 5.66
TESTING:
Accuracy of the network on the 10000 test images: 51.69 %
Average loss on the 10000 test images: 1.352
[10,   100] loss: 1.371 acc: 50.27 time: 5.91
[10,   200] loss: 1.359 acc: 51.35 time: 7.36
[10,   300] loss: 1.369 acc: 51.49 time: 5.54
TESTING:
Accuracy of the network on the 10000 test images: 52.97 %
Average loss on the 10000 test images: 1.311
[11,   100] loss: 1.297 acc: 54.16 time: 5.73
[11,   200] loss: 1.272 acc: 54.60 time: 7.30
[11,   300] loss: 1.262 acc: 54.52 time: 5.54
TESTING:
Accuracy of the network on the 10000 test images: 56.05 %
Average loss on the 10000 test images: 1.226
[12,   100] loss: 1.251 acc: 55.45 time: 5.83
[12,   200] loss: 1.271 acc: 54.39 time: 7.12
[12,   300] loss: 1.250 acc: 55.32 time: 5.64
TESTING:
Accuracy of the network on the 10000 test images: 56.31 %
Average loss on the 10000 test images: 1.217
[13,   100] loss: 1.257 acc: 54.70 time: 5.90
[13,   200] loss: 1.248 acc: 55.02 time: 7.34
[13,   300] loss: 1.249 acc: 55.13 time: 5.56
TESTING:
Accuracy of the network on the 10000 test images: 56.72 %
Average loss on the 10000 test images: 1.208
```

```
[14,   100] loss: 1.245 acc: 54.97 time: 5.84
[14,   200] loss: 1.248 acc: 55.63 time: 7.39
[14,   300] loss: 1.246 acc: 55.11 time: 5.49
TESTING:
Accuracy of the network on the 10000 test images: 56.53 %
Average loss on the 10000 test images: 1.210
[15,   100] loss: 1.231 acc: 55.73 time: 5.70
[15,   200] loss: 1.251 acc: 54.80 time: 7.38
[15,   300] loss: 1.235 acc: 55.70 time: 5.58
TESTING:
Accuracy of the network on the 10000 test images: 56.11 %
Average loss on the 10000 test images: 1.209
[16,   100] loss: 1.252 acc: 55.11 time: 5.76
[16,   200] loss: 1.255 acc: 54.66 time: 7.38
[16,   300] loss: 1.225 acc: 55.93 time: 5.59
TESTING:
Accuracy of the network on the 10000 test images: 56.91 %
Average loss on the 10000 test images: 1.198
[17,   100] loss: 1.239 acc: 55.78 time: 5.72
[17,   200] loss: 1.221 acc: 56.00 time: 7.50
[17,   300] loss: 1.247 acc: 54.94 time: 5.53
TESTING:
Accuracy of the network on the 10000 test images: 56.89 %
Average loss on the 10000 test images: 1.202
[18,   100] loss: 1.232 acc: 55.38 time: 5.60
[18,   200] loss: 1.227 acc: 55.91 time: 7.52
[18,   300] loss: 1.231 acc: 55.51 time: 5.67
TESTING:
Accuracy of the network on the 10000 test images: 56.84 %
Average loss on the 10000 test images: 1.192
[19,   100] loss: 1.226 acc: 56.26 time: 5.77
[19,   200] loss: 1.235 acc: 55.52 time: 7.17
[19,   300] loss: 1.226 acc: 55.84 time: 5.57
TESTING:
Accuracy of the network on the 10000 test images: 56.65 %
Average loss on the 10000 test images: 1.198
[20,   100] loss: 1.224 acc: 56.34 time: 5.79
[20,   200] loss: 1.222 acc: 56.09 time: 7.17
[20,   300] loss: 1.226 acc: 55.86 time: 5.77
TESTING:
Accuracy of the network on the 10000 test images: 56.82 %
Average loss on the 10000 test images: 1.188
[21,   100] loss: 1.239 acc: 56.26 time: 5.83
[21,   200] loss: 1.210 acc: 56.09 time: 7.38
[21,   300] loss: 1.218 acc: 56.16 time: 5.59
TESTING:
Accuracy of the network on the 10000 test images: 57.04 %
Average loss on the 10000 test images: 1.182
```

```
[22,   100] loss: 1.221 acc: 56.25 time: 5.70
[22,   200] loss: 1.211 acc: 56.27 time: 7.26
[22,   300] loss: 1.219 acc: 56.41 time: 5.56
TESTING:
Accuracy of the network on the 10000 test images: 57.42 %
Average loss on the 10000 test images: 1.180
[23,   100] loss: 1.206 acc: 57.04 time: 5.66
[23,   200] loss: 1.207 acc: 56.38 time: 7.24
[23,   300] loss: 1.212 acc: 56.41 time: 5.52
TESTING:
Accuracy of the network on the 10000 test images: 57.47 %
Average loss on the 10000 test images: 1.181
[24,   100] loss: 1.204 acc: 55.89 time: 5.75
[24,   200] loss: 1.213 acc: 56.26 time: 7.30
[24,   300] loss: 1.202 acc: 56.57 time: 5.60
TESTING:
Accuracy of the network on the 10000 test images: 57.39 %
Average loss on the 10000 test images: 1.178
[25,   100] loss: 1.212 acc: 56.20 time: 5.70
[25,   200] loss: 1.224 acc: 55.50 time: 7.31
[25,   300] loss: 1.208 acc: 56.18 time: 5.50
TESTING:
Accuracy of the network on the 10000 test images: 57.21 %
Average loss on the 10000 test images: 1.182
[26,   100] loss: 1.211 acc: 56.42 time: 5.75
[26,   200] loss: 1.206 acc: 56.30 time: 7.42
[26,   300] loss: 1.216 acc: 56.52 time: 5.45
TESTING:
Accuracy of the network on the 10000 test images: 57.29 %
Average loss on the 10000 test images: 1.185
[27,   100] loss: 1.200 acc: 56.34 time: 5.92
[27,   200] loss: 1.195 acc: 57.05 time: 7.13
[27,   300] loss: 1.228 acc: 56.31 time: 5.66
TESTING:
Accuracy of the network on the 10000 test images: 57.56 %
Average loss on the 10000 test images: 1.178
[28,   100] loss: 1.219 acc: 55.80 time: 6.28
[28,   200] loss: 1.210 acc: 56.52 time: 6.98
[28,   300] loss: 1.216 acc: 55.98 time: 5.57
TESTING:
Accuracy of the network on the 10000 test images: 57.30 %
Average loss on the 10000 test images: 1.179
[29,   100] loss: 1.218 acc: 55.80 time: 6.18
[29,   200] loss: 1.217 acc: 56.33 time: 6.90
[29,   300] loss: 1.206 acc: 56.64 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 57.38 %
Average loss on the 10000 test images: 1.182
```

```
[30,   100] loss: 1.224 acc: 56.13 time: 6.08
[30,   200] loss: 1.209 acc: 55.90 time: 7.06
[30,   300] loss: 1.213 acc: 56.25 time: 5.52
TESTING:
Accuracy of the network on the 10000 test images: 57.34 %
Average loss on the 10000 test images: 1.181
Finished Training
```

1.1 Fine-tuning on the randomly initialized model

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
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import resnet18

# Randomly initialize a ResNet18 model
net = resnet18(num_classes=10)

net = net.to(device)
```

[ ]:
```python
for param in net.parameters():
    param.requires_grad = False

# Unfreeze the 'layer4' block and 'fc' layer
for param in net.layer4.parameters():
    param.requires_grad = True

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

[ ]:
```python
# 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
```

[ ]:
```
# criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)

criterion = nn.CrossEntropyLoss()
#optimizer = optim.Adam(params_to_update, lr=0.001, weight_decay=1e-2)
```

[ ]:
```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=4, init_lr=0.1,␣
 ↪task='classification')
```

```
[1,   100] loss: 3.534 acc: 21.99 time: 5.90
[1,   200] loss: 2.007 acc: 29.26 time: 7.46
[1,   300] loss: 1.884 acc: 31.75 time: 5.69
TESTING:
Accuracy of the network on the 10000 test images: 37.71 %
Average loss on the 10000 test images: 1.738
[2,   100] loss: 1.815 acc: 33.91 time: 5.96
[2,   200] loss: 1.791 acc: 34.80 time: 8.65
[2,   300] loss: 1.776 acc: 36.28 time: 5.91
TESTING:
Accuracy of the network on the 10000 test images: 38.72 %
Average loss on the 10000 test images: 1.694
[3,   100] loss: 1.748 acc: 36.70 time: 7.13
[3,   200] loss: 1.742 acc: 36.58 time: 6.13
[3,   300] loss: 1.740 acc: 37.04 time: 6.12
TESTING:
Accuracy of the network on the 10000 test images: 39.48 %
Average loss on the 10000 test images: 1.661
[4,   100] loss: 1.715 acc: 37.73 time: 7.58
[4,   200] loss: 1.719 acc: 38.31 time: 5.82
[4,   300] loss: 1.707 acc: 38.20 time: 6.90
TESTING:
Accuracy of the network on the 10000 test images: 39.20 %
Average loss on the 10000 test images: 1.672
[5,   100] loss: 1.684 acc: 38.98 time: 7.68
[5,   200] loss: 1.644 acc: 40.26 time: 5.74
```

```
[5,   300] loss: 1.648 acc: 40.50 time: 7.32
TESTING:
Accuracy of the network on the 10000 test images: 43.12 %
Average loss on the 10000 test images: 1.577
[6,   100] loss: 1.641 acc: 41.09 time: 7.80
[6,   200] loss: 1.627 acc: 41.57 time: 5.74
[6,   300] loss: 1.615 acc: 42.73 time: 7.58
TESTING:
Accuracy of the network on the 10000 test images: 43.42 %
Average loss on the 10000 test images: 1.570
[7,   100] loss: 1.610 acc: 41.93 time: 7.81
[7,   200] loss: 1.616 acc: 41.75 time: 5.68
[7,   300] loss: 1.624 acc: 42.16 time: 7.63
TESTING:
Accuracy of the network on the 10000 test images: 44.03 %
Average loss on the 10000 test images: 1.559
[8,   100] loss: 1.613 acc: 42.23 time: 7.29
[8,   200] loss: 1.617 acc: 41.77 time: 5.72
[8,   300] loss: 1.613 acc: 41.92 time: 7.40
TESTING:
Accuracy of the network on the 10000 test images: 44.13 %
Average loss on the 10000 test images: 1.550
[9,   100] loss: 1.597 acc: 42.89 time: 6.74
[9,   200] loss: 1.604 acc: 42.06 time: 5.79
[9,   300] loss: 1.605 acc: 42.11 time: 7.40
TESTING:
Accuracy of the network on the 10000 test images: 44.30 %
Average loss on the 10000 test images: 1.542
[10,   100] loss: 1.587 acc: 43.25 time: 6.28
[10,   200] loss: 1.604 acc: 42.30 time: 6.50
[10,   300] loss: 1.600 acc: 42.35 time: 6.79
TESTING:
Accuracy of the network on the 10000 test images: 44.51 %
Average loss on the 10000 test images: 1.541
[11,   100] loss: 1.595 acc: 42.76 time: 7.73
[11,   200] loss: 1.589 acc: 42.77 time: 8.31
[11,   300] loss: 1.587 acc: 43.59 time: 6.62
TESTING:
Accuracy of the network on the 10000 test images: 44.48 %
Average loss on the 10000 test images: 1.542
[12,   100] loss: 1.580 acc: 43.07 time: 7.60
[12,   200] loss: 1.597 acc: 42.97 time: 5.80
[12,   300] loss: 1.596 acc: 42.87 time: 7.35
TESTING:
Accuracy of the network on the 10000 test images: 44.64 %
Average loss on the 10000 test images: 1.540
[13,   100] loss: 1.594 acc: 42.87 time: 7.57
[13,   200] loss: 1.582 acc: 42.91 time: 5.70
```

```
[13,   300] loss: 1.585 acc: 43.18 time: 7.51
TESTING:
Accuracy of the network on the 10000 test images: 44.62 %
Average loss on the 10000 test images: 1.539
[14,   100] loss: 1.592 acc: 42.90 time: 7.55
[14,   200] loss: 1.591 acc: 43.29 time: 5.73
[14,   300] loss: 1.595 acc: 42.69 time: 7.41
TESTING:
Accuracy of the network on the 10000 test images: 44.53 %
Average loss on the 10000 test images: 1.540
[15,   100] loss: 1.602 acc: 42.25 time: 7.57
[15,   200] loss: 1.592 acc: 42.96 time: 5.75
[15,   300] loss: 1.594 acc: 42.89 time: 7.33
TESTING:
Accuracy of the network on the 10000 test images: 44.58 %
Average loss on the 10000 test images: 1.539
[16,   100] loss: 1.591 acc: 43.20 time: 7.42
[16,   200] loss: 1.592 acc: 43.05 time: 5.74
[16,   300] loss: 1.590 acc: 43.02 time: 7.49
TESTING:
Accuracy of the network on the 10000 test images: 44.78 %
Average loss on the 10000 test images: 1.538
[17,   100] loss: 1.584 acc: 42.88 time: 6.63
[17,   200] loss: 1.590 acc: 42.67 time: 5.91
[17,   300] loss: 1.600 acc: 43.13 time: 7.15
TESTING:
Accuracy of the network on the 10000 test images: 44.52 %
Average loss on the 10000 test images: 1.540
[18,   100] loss: 1.579 acc: 43.56 time: 6.41
[18,   200] loss: 1.592 acc: 42.66 time: 6.33
[18,   300] loss: 1.592 acc: 42.36 time: 6.97
TESTING:
Accuracy of the network on the 10000 test images: 44.72 %
Average loss on the 10000 test images: 1.540
[19,   100] loss: 1.588 acc: 42.83 time: 5.78
[19,   200] loss: 1.601 acc: 42.36 time: 6.68
[19,   300] loss: 1.588 acc: 43.06 time: 6.50
TESTING:
Accuracy of the network on the 10000 test images: 44.51 %
Average loss on the 10000 test images: 1.541
[20,   100] loss: 1.585 acc: 43.10 time: 5.82
[20,   200] loss: 1.594 acc: 42.66 time: 7.22
[20,   300] loss: 1.587 acc: 43.18 time: 5.97
TESTING:
Accuracy of the network on the 10000 test images: 44.53 %
Average loss on the 10000 test images: 1.539
Finished Training
```

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

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

from torchvision.models import resnet18

net = models.resnet18(num_classes=4)

saved_path = 'rotation_model.pth'
net.load_state_dict(torch.load(saved_path))

# Move the model to the device (GPU or CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
net.to(device)

num_classes = 10
net.fc = nn.Linear(net.fc.in_features, num_classes).to(device)
```

[ ]:
```
optimizer = optim.SGD(net.parameters(), lr=0.01, weight_decay=1e-2)

criterion = nn.CrossEntropyLoss()
#optimizer = optim.Adam(net.parameters(), lr=0.1)
```

[ ]:
```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=4, init_lr=0.1,␣
↪task='classification')
```

```
[1,   100] loss: 0.775 acc: 73.11 time: 6.02
[1,   200] loss: 0.773 acc: 72.98 time: 7.19
[1,   300] loss: 0.767 acc: 73.49 time: 5.92
TESTING:
Accuracy of the network on the 10000 test images: 73.87 %
Average loss on the 10000 test images: 0.754
[2,   100] loss: 0.777 acc: 72.91 time: 5.85
[2,   200] loss: 0.769 acc: 73.43 time: 7.41
[2,   300] loss: 0.764 acc: 73.38 time: 5.81
TESTING:
Accuracy of the network on the 10000 test images: 72.19 %
Average loss on the 10000 test images: 0.801
[3,   100] loss: 0.782 acc: 73.09 time: 6.03
[3,   200] loss: 0.785 acc: 72.61 time: 7.26
[3,   300] loss: 0.789 acc: 72.83 time: 5.77
TESTING:
Accuracy of the network on the 10000 test images: 71.88 %
Average loss on the 10000 test images: 0.817
[4,   100] loss: 0.806 acc: 72.06 time: 6.35
[4,   200] loss: 0.806 acc: 72.11 time: 8.15
```

```
[4,   300] loss: 0.822 acc: 71.71 time: 5.88
TESTING:
Accuracy of the network on the 10000 test images: 67.50 %
Average loss on the 10000 test images: 0.920
[5,   100] loss: 0.760 acc: 74.11 time: 6.92
[5,   200] loss: 0.760 acc: 74.28 time: 6.60
[5,   300] loss: 0.738 acc: 74.94 time: 6.45
TESTING:
Accuracy of the network on the 10000 test images: 75.37 %
Average loss on the 10000 test images: 0.714
[6,   100] loss: 0.725 acc: 75.42 time: 7.61
[6,   200] loss: 0.743 acc: 74.50 time: 5.76
[6,   300] loss: 0.730 acc: 75.02 time: 7.47
TESTING:
Accuracy of the network on the 10000 test images: 75.69 %
Average loss on the 10000 test images: 0.713
[7,   100] loss: 0.734 acc: 74.70 time: 7.61
[7,   200] loss: 0.724 acc: 75.29 time: 5.83
[7,   300] loss: 0.725 acc: 75.02 time: 7.42
TESTING:
Accuracy of the network on the 10000 test images: 75.49 %
Average loss on the 10000 test images: 0.702
[8,   100] loss: 0.728 acc: 75.30 time: 7.66
[8,   200] loss: 0.715 acc: 75.68 time: 5.84
[8,   300] loss: 0.731 acc: 74.98 time: 7.40
TESTING:
Accuracy of the network on the 10000 test images: 75.91 %
Average loss on the 10000 test images: 0.700
[9,   100] loss: 0.710 acc: 75.83 time: 6.39
[9,   200] loss: 0.710 acc: 75.42 time: 6.55
[9,   300] loss: 0.711 acc: 74.98 time: 6.69
TESTING:
Accuracy of the network on the 10000 test images: 76.16 %
Average loss on the 10000 test images: 0.692
[10,   100] loss: 0.718 acc: 75.62 time: 5.99
[10,   200] loss: 0.717 acc: 75.31 time: 7.55
[10,   300] loss: 0.709 acc: 75.80 time: 5.88
TESTING:
Accuracy of the network on the 10000 test images: 76.23 %
Average loss on the 10000 test images: 0.690
[11,   100] loss: 0.718 acc: 75.23 time: 6.15
[11,   200] loss: 0.721 acc: 75.44 time: 7.47
[11,   300] loss: 0.695 acc: 76.09 time: 5.86
TESTING:
Accuracy of the network on the 10000 test images: 76.24 %
Average loss on the 10000 test images: 0.689
[12,   100] loss: 0.714 acc: 75.37 time: 5.98
[12,   200] loss: 0.710 acc: 75.74 time: 7.56
```

```
[12,   300] loss: 0.701 acc: 75.95 time: 5.79
TESTING:
Accuracy of the network on the 10000 test images: 76.21 %
Average loss on the 10000 test images: 0.689
[13,   100] loss: 0.719 acc: 75.58 time: 7.22
[13,   200] loss: 0.713 acc: 75.85 time: 6.35
[13,   300] loss: 0.704 acc: 76.05 time: 6.53
TESTING:
Accuracy of the network on the 10000 test images: 76.14 %
Average loss on the 10000 test images: 0.689
[14,   100] loss: 0.708 acc: 75.77 time: 7.62
[14,   200] loss: 0.700 acc: 76.07 time: 5.78
[14,   300] loss: 0.710 acc: 75.56 time: 7.19
TESTING:
Accuracy of the network on the 10000 test images: 76.27 %
Average loss on the 10000 test images: 0.689
[15,   100] loss: 0.701 acc: 75.91 time: 7.61
[15,   200] loss: 0.705 acc: 75.99 time: 5.79
[15,   300] loss: 0.707 acc: 75.82 time: 7.50
TESTING:
Accuracy of the network on the 10000 test images: 76.10 %
Average loss on the 10000 test images: 0.690
[16,   100] loss: 0.717 acc: 75.63 time: 8.00
[16,   200] loss: 0.711 acc: 75.48 time: 6.00
[16,   300] loss: 0.690 acc: 76.33 time: 7.41
TESTING:
Accuracy of the network on the 10000 test images: 76.30 %
Average loss on the 10000 test images: 0.690
[17,   100] loss: 0.703 acc: 75.72 time: 6.59
[17,   200] loss: 0.713 acc: 75.69 time: 6.54
[17,   300] loss: 0.697 acc: 75.80 time: 6.68
TESTING:
Accuracy of the network on the 10000 test images: 76.31 %
Average loss on the 10000 test images: 0.687
[18,   100] loss: 0.714 acc: 75.35 time: 6.01
[18,   200] loss: 0.695 acc: 76.35 time: 7.38
[18,   300] loss: 0.717 acc: 75.35 time: 5.86
TESTING:
Accuracy of the network on the 10000 test images: 76.37 %
Average loss on the 10000 test images: 0.688
[19,   100] loss: 0.698 acc: 76.02 time: 6.01
[19,   200] loss: 0.700 acc: 76.08 time: 7.50
[19,   300] loss: 0.703 acc: 75.93 time: 5.80
TESTING:
Accuracy of the network on the 10000 test images: 76.38 %
Average loss on the 10000 test images: 0.688
[20,   100] loss: 0.699 acc: 76.04 time: 5.99
[20,   200] loss: 0.708 acc: 75.82 time: 7.47
```

```
[20,   300] loss: 0.693 acc: 76.72 time: 5.80
TESTING:
Accuracy of the network on the 10000 test images: 76.31 %
Average loss on the 10000 test images: 0.689
Finished Training
```

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

[ ]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import resnet18

# Randomly initialize a ResNet18 model
net = resnet18(num_classes=10)

net = net.to(device)
```

[ ]:
```python
# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
```

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

```
[1,   100] loss: 2.046 acc: 26.81 time: 13.62
[1,   200] loss: 1.719 acc: 37.32 time: 8.32
[1,   300] loss: 1.574 acc: 42.45 time: 6.32
TESTING:
Accuracy of the network on the 10000 test images: 50.10 %
Average loss on the 10000 test images: 1.395
[2,   100] loss: 1.444 acc: 47.64 time: 6.02
[2,   200] loss: 1.374 acc: 50.68 time: 7.47
[2,   300] loss: 1.304 acc: 53.44 time: 5.87
TESTING:
Accuracy of the network on the 10000 test images: 55.24 %
Average loss on the 10000 test images: 1.267
[3,   100] loss: 1.229 acc: 55.73 time: 8.78
[3,   200] loss: 1.189 acc: 57.82 time: 10.25
[3,   300] loss: 1.165 acc: 57.84 time: 9.39
TESTING:
Accuracy of the network on the 10000 test images: 58.02 %
Average loss on the 10000 test images: 1.218
[4,   100] loss: 1.101 acc: 61.19 time: 7.15
[4,   200] loss: 1.101 acc: 60.45 time: 6.65
[4,   300] loss: 1.059 acc: 62.81 time: 7.44
TESTING:
```

```
Accuracy of the network on the 10000 test images: 63.80 %
Average loss on the 10000 test images: 1.060
[5,   100] loss: 0.998 acc: 64.66 time: 6.13
[5,   200] loss: 1.002 acc: 64.44 time: 7.72
[5,   300] loss: 0.976 acc: 65.42 time: 10.47
TESTING:
Accuracy of the network on the 10000 test images: 68.51 %
Average loss on the 10000 test images: 0.894
[6,   100] loss: 0.918 acc: 67.23 time: 7.63
[6,   200] loss: 0.923 acc: 67.88 time: 5.98
[6,   300] loss: 0.903 acc: 68.60 time: 6.92
TESTING:
Accuracy of the network on the 10000 test images: 69.62 %
Average loss on the 10000 test images: 0.877
[7,   100] loss: 0.865 acc: 69.66 time: 7.70
[7,   200] loss: 0.865 acc: 69.45 time: 6.00
[7,   300] loss: 0.850 acc: 70.41 time: 7.69
TESTING:
Accuracy of the network on the 10000 test images: 69.53 %
Average loss on the 10000 test images: 0.860
[8,   100] loss: 0.800 acc: 71.80 time: 8.48
[8,   200] loss: 0.839 acc: 70.52 time: 6.12
[8,   300] loss: 0.807 acc: 71.52 time: 7.64
TESTING:
Accuracy of the network on the 10000 test images: 70.81 %
Average loss on the 10000 test images: 0.837
[9,   100] loss: 0.773 acc: 72.77 time: 7.37
[9,   200] loss: 0.779 acc: 72.39 time: 5.95
[9,   300] loss: 0.775 acc: 72.98 time: 7.59
TESTING:
Accuracy of the network on the 10000 test images: 73.32 %
Average loss on the 10000 test images: 0.765
[10,   100] loss: 0.746 acc: 73.81 time: 6.56
[10,   200] loss: 0.742 acc: 74.12 time: 6.96
[10,   300] loss: 0.750 acc: 73.50 time: 8.38
TESTING:
Accuracy of the network on the 10000 test images: 72.40 %
Average loss on the 10000 test images: 0.810
[11,   100] loss: 0.661 acc: 76.93 time: 6.25
[11,   200] loss: 0.613 acc: 78.48 time: 8.72
[11,   300] loss: 0.619 acc: 77.93 time: 5.98
TESTING:
Accuracy of the network on the 10000 test images: 77.37 %
Average loss on the 10000 test images: 0.645
[12,   100] loss: 0.598 acc: 78.93 time: 8.27
[12,   200] loss: 0.603 acc: 78.67 time: 7.26
[12,   300] loss: 0.594 acc: 79.13 time: 7.20
TESTING:
```

```
Accuracy of the network on the 10000 test images: 77.71 %
Average loss on the 10000 test images: 0.638
[13,   100] loss: 0.583 acc: 79.32 time: 9.44
[13,   200] loss: 0.584 acc: 79.28 time: 6.00
[13,   300] loss: 0.587 acc: 79.33 time: 7.60
TESTING:
Accuracy of the network on the 10000 test images: 77.92 %
Average loss on the 10000 test images: 0.627
[14,   100] loss: 0.563 acc: 80.37 time: 6.93
[14,   200] loss: 0.561 acc: 80.04 time: 6.20
[14,   300] loss: 0.586 acc: 79.17 time: 8.57
TESTING:
Accuracy of the network on the 10000 test images: 78.11 %
Average loss on the 10000 test images: 0.620
[15,   100] loss: 0.551 acc: 80.55 time: 6.01
[15,   200] loss: 0.572 acc: 79.56 time: 8.62
[15,   300] loss: 0.566 acc: 80.03 time: 6.78
TESTING:
Accuracy of the network on the 10000 test images: 78.17 %
Average loss on the 10000 test images: 0.621
[16,   100] loss: 0.559 acc: 80.12 time: 6.59
[16,   200] loss: 0.553 acc: 80.48 time: 8.32
[16,   300] loss: 0.554 acc: 80.65 time: 6.91
TESTING:
Accuracy of the network on the 10000 test images: 78.28 %
Average loss on the 10000 test images: 0.618
[17,   100] loss: 0.537 acc: 81.26 time: 7.69
[17,   200] loss: 0.546 acc: 80.70 time: 5.89
[17,   300] loss: 0.542 acc: 81.01 time: 7.47
TESTING:
Accuracy of the network on the 10000 test images: 78.45 %
Average loss on the 10000 test images: 0.615
[18,   100] loss: 0.537 acc: 81.40 time: 8.00
[18,   200] loss: 0.537 acc: 80.86 time: 6.81
[18,   300] loss: 0.533 acc: 81.22 time: 7.71
TESTING:
Accuracy of the network on the 10000 test images: 78.49 %
Average loss on the 10000 test images: 0.610
[19,   100] loss: 0.538 acc: 81.20 time: 6.49
[19,   200] loss: 0.529 acc: 81.20 time: 7.43
[19,   300] loss: 0.537 acc: 80.57 time: 6.33
TESTING:
Accuracy of the network on the 10000 test images: 78.82 %
Average loss on the 10000 test images: 0.606
[20,   100] loss: 0.521 acc: 81.38 time: 6.15
[20,   200] loss: 0.532 acc: 81.63 time: 7.59
[20,   300] loss: 0.528 acc: 81.49 time: 5.84
TESTING:
```

```
Accuracy of the network on the 10000 test images: 78.81 %
Average loss on the 10000 test images: 0.602
Finished Training
```

Extra part a)

```
[ ]:
```

Part 1 Extra Credit (b)

```
[ ]: # Load the pre-trained ResNet101 model
     resnet50 = models.resnet50(pretrained=True)

     # Modify the last layer for rotation prediction task
     num_features = resnet50.fc.in_features
     resnet50.fc = nn.Linear(num_features, 4)
```

```
/usr/local/lib/python3.9/dist-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(
/usr/local/lib/python3.9/dist-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=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|      | 97.8M/97.8M [00:00<00:00, 223MB/s]
```

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

     device
```

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

```
[ ]: resnet50 = resnet50.to(device)
```

```
[ ]: import torch.optim as optim
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(resnet50.parameters(), lr=0.1, momentum=0.9,␣
      ↪weight_decay=1e-2)
```

```
[ ]: train(resnet50, criterion, optimizer, num_epochs=45, decay_epochs=5, init_lr=0.
      ↪01, task='rotation')
```

```
[1,   100] loss: 0.980 acc: 57.22 time: 9.06
[1,   200] loss: 0.655 acc: 71.86 time: 7.49
```

```
[1,   300] loss: 0.579 acc: 75.28 time: 8.76
TESTING:
Accuracy of the network on the 10000 test images: 74.12 %
Average loss on the 10000 test images: 0.659
[2,   100] loss: 0.517 acc: 78.23 time: 7.95
[2,   200] loss: 0.499 acc: 79.44 time: 8.57
[2,   300] loss: 0.494 acc: 79.49 time: 8.75
TESTING:
Accuracy of the network on the 10000 test images: 71.96 %
Average loss on the 10000 test images: 0.711
[3,   100] loss: 0.509 acc: 79.09 time: 8.88
[3,   200] loss: 0.516 acc: 78.77 time: 8.88
[3,   300] loss: 0.525 acc: 77.98 time: 7.44
TESTING:
Accuracy of the network on the 10000 test images: 69.65 %
Average loss on the 10000 test images: 0.788
[4,   100] loss: 0.532 acc: 77.73 time: 8.96
[4,   200] loss: 0.543 acc: 77.07 time: 7.55
[4,   300] loss: 0.557 acc: 77.02 time: 8.84
TESTING:
Accuracy of the network on the 10000 test images: 66.29 %
Average loss on the 10000 test images: 0.839
[5,   100] loss: 0.544 acc: 77.17 time: 7.64
[5,   200] loss: 0.569 acc: 76.21 time: 8.82
[5,   300] loss: 0.578 acc: 75.64 time: 7.79
TESTING:
Accuracy of the network on the 10000 test images: 63.49 %
Average loss on the 10000 test images: 0.900
[6,   100] loss: 0.493 acc: 79.46 time: 9.15
[6,   200] loss: 0.433 acc: 82.34 time: 7.81
[6,   300] loss: 0.420 acc: 82.73 time: 8.56
TESTING:
Accuracy of the network on the 10000 test images: 78.86 %
Average loss on the 10000 test images: 0.555
[7,   100] loss: 0.387 acc: 84.32 time: 7.96
[7,   200] loss: 0.391 acc: 83.94 time: 8.55
[7,   300] loss: 0.393 acc: 84.23 time: 9.24
TESTING:
Accuracy of the network on the 10000 test images: 80.01 %
Average loss on the 10000 test images: 0.525
[8,   100] loss: 0.373 acc: 85.06 time: 8.79
[8,   200] loss: 0.372 acc: 85.31 time: 8.77
[8,   300] loss: 0.367 acc: 85.25 time: 7.53
TESTING:
Accuracy of the network on the 10000 test images: 80.53 %
Average loss on the 10000 test images: 0.510
[9,   100] loss: 0.357 acc: 85.50 time: 8.95
[9,   200] loss: 0.361 acc: 85.62 time: 7.47
```

```
[9,   300] loss: 0.371 acc: 85.05 time: 8.84
TESTING:
Accuracy of the network on the 10000 test images: 81.14 %
Average loss on the 10000 test images: 0.508
[10,   100] loss: 0.350 acc: 85.88 time: 7.74
[10,   200] loss: 0.341 acc: 86.41 time: 8.76
[10,   300] loss: 0.352 acc: 85.62 time: 7.80
TESTING:
Accuracy of the network on the 10000 test images: 81.13 %
Average loss on the 10000 test images: 0.493
[11,   100] loss: 0.327 acc: 87.20 time: 9.06
[11,   200] loss: 0.327 acc: 87.02 time: 7.90
[11,   300] loss: 0.326 acc: 87.02 time: 8.43
TESTING:
Accuracy of the network on the 10000 test images: 83.53 %
Average loss on the 10000 test images: 0.445
[12,   100] loss: 0.312 acc: 87.95 time: 8.07
[12,   200] loss: 0.302 acc: 87.87 time: 8.50
[12,   300] loss: 0.305 acc: 87.92 time: 8.87
TESTING:
Accuracy of the network on the 10000 test images: 83.95 %
Average loss on the 10000 test images: 0.436
[13,   100] loss: 0.303 acc: 88.12 time: 8.89
[13,   200] loss: 0.313 acc: 87.80 time: 8.67
[13,   300] loss: 0.299 acc: 88.41 time: 7.45
TESTING:
Accuracy of the network on the 10000 test images: 83.61 %
Average loss on the 10000 test images: 0.439
[14,   100] loss: 0.293 acc: 88.53 time: 8.80
[14,   200] loss: 0.296 acc: 88.58 time: 7.63
[14,   300] loss: 0.298 acc: 88.66 time: 8.90
TESTING:
Accuracy of the network on the 10000 test images: 84.09 %
Average loss on the 10000 test images: 0.425
[15,   100] loss: 0.291 acc: 88.68 time: 7.80
[15,   200] loss: 0.287 acc: 89.23 time: 8.91
[15,   300] loss: 0.299 acc: 88.32 time: 7.55
TESTING:
Accuracy of the network on the 10000 test images: 84.43 %
Average loss on the 10000 test images: 0.418
[16,   100] loss: 0.297 acc: 88.20 time: 8.95
[16,   200] loss: 0.291 acc: 88.73 time: 7.56
[16,   300] loss: 0.282 acc: 89.31 time: 8.72
TESTING:
Accuracy of the network on the 10000 test images: 84.60 %
Average loss on the 10000 test images: 0.418
[17,   100] loss: 0.282 acc: 89.40 time: 7.70
[17,   200] loss: 0.293 acc: 88.34 time: 8.76
```

```
[17,   300] loss: 0.285 acc: 89.23 time: 8.55
TESTING:
Accuracy of the network on the 10000 test images: 84.22 %
Average loss on the 10000 test images: 0.419
[18,   100] loss: 0.299 acc: 88.44 time: 9.10
[18,   200] loss: 0.284 acc: 88.84 time: 8.34
[18,   300] loss: 0.288 acc: 88.96 time: 7.92
TESTING:
Accuracy of the network on the 10000 test images: 84.51 %
Average loss on the 10000 test images: 0.412
[19,   100] loss: 0.282 acc: 89.16 time: 8.52
[19,   200] loss: 0.288 acc: 89.01 time: 8.49
[19,   300] loss: 0.298 acc: 88.28 time: 9.17
TESTING:
Accuracy of the network on the 10000 test images: 84.68 %
Average loss on the 10000 test images: 0.413
[20,   100] loss: 0.289 acc: 89.19 time: 8.27
[20,   200] loss: 0.282 acc: 88.96 time: 9.11
[20,   300] loss: 0.293 acc: 89.01 time: 7.47
TESTING:
Accuracy of the network on the 10000 test images: 84.07 %
Average loss on the 10000 test images: 0.421
[21,   100] loss: 0.294 acc: 88.58 time: 9.12
[21,   200] loss: 0.287 acc: 88.74 time: 7.21
[21,   300] loss: 0.285 acc: 88.91 time: 8.81
TESTING:
Accuracy of the network on the 10000 test images: 84.70 %
Average loss on the 10000 test images: 0.412
[22,   100] loss: 0.293 acc: 88.48 time: 7.68
[22,   200] loss: 0.293 acc: 88.73 time: 8.95
[22,   300] loss: 0.287 acc: 88.82 time: 8.50
TESTING:
Accuracy of the network on the 10000 test images: 84.54 %
Average loss on the 10000 test images: 0.417
[23,   100] loss: 0.283 acc: 89.11 time: 9.25
[23,   200] loss: 0.282 acc: 88.99 time: 8.48
[23,   300] loss: 0.288 acc: 88.63 time: 7.98
TESTING:
Accuracy of the network on the 10000 test images: 85.08 %
Average loss on the 10000 test images: 0.410
[24,   100] loss: 0.281 acc: 89.23 time: 8.74
[24,   200] loss: 0.300 acc: 88.36 time: 7.97
[24,   300] loss: 0.295 acc: 88.47 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 84.45 %
Average loss on the 10000 test images: 0.422
[25,   100] loss: 0.287 acc: 88.80 time: 8.20
[25,   200] loss: 0.289 acc: 88.70 time: 8.99
```

```
[25,   300] loss: 0.284 acc: 89.03 time: 7.53
TESTING:
Accuracy of the network on the 10000 test images: 84.34 %
Average loss on the 10000 test images: 0.419
[26,   100] loss: 0.293 acc: 88.53 time: 9.02
[26,   200] loss: 0.293 acc: 88.55 time: 7.51
[26,   300] loss: 0.289 acc: 89.05 time: 8.79
TESTING:
Accuracy of the network on the 10000 test images: 84.88 %
Average loss on the 10000 test images: 0.408
[27,   100] loss: 0.288 acc: 89.08 time: 7.71
[27,   200] loss: 0.285 acc: 89.00 time: 8.93
[27,   300] loss: 0.287 acc: 88.86 time: 8.49
TESTING:
Accuracy of the network on the 10000 test images: 84.83 %
Average loss on the 10000 test images: 0.414
[28,   100] loss: 0.294 acc: 88.63 time: 9.16
[28,   200] loss: 0.296 acc: 88.45 time: 8.49
[28,   300] loss: 0.281 acc: 89.32 time: 7.75
TESTING:
Accuracy of the network on the 10000 test images: 84.33 %
Average loss on the 10000 test images: 0.424
[29,   100] loss: 0.287 acc: 89.03 time: 8.77
[29,   200] loss: 0.283 acc: 89.36 time: 7.95
[29,   300] loss: 0.295 acc: 88.49 time: 8.78
TESTING:
Accuracy of the network on the 10000 test images: 84.24 %
Average loss on the 10000 test images: 0.419
[30,   100] loss: 0.280 acc: 89.31 time: 8.08
[30,   200] loss: 0.285 acc: 89.16 time: 8.79
[30,   300] loss: 0.296 acc: 88.36 time: 7.55
TESTING:
Accuracy of the network on the 10000 test images: 84.46 %
Average loss on the 10000 test images: 0.420
[31,   100] loss: 0.290 acc: 88.62 time: 9.30
[31,   200] loss: 0.285 acc: 89.15 time: 7.61
[31,   300] loss: 0.287 acc: 88.62 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 85.04 %
Average loss on the 10000 test images: 0.410
[32,   100] loss: 0.288 acc: 88.80 time: 7.79
[32,   200] loss: 0.292 acc: 88.70 time: 8.73
[32,   300] loss: 0.282 acc: 89.26 time: 8.69
TESTING:
Accuracy of the network on the 10000 test images: 84.81 %
Average loss on the 10000 test images: 0.411
[33,   100] loss: 0.294 acc: 88.41 time: 9.17
[33,   200] loss: 0.290 acc: 88.93 time: 9.18
```

```
[33,   300] loss: 0.293 acc: 88.76 time: 7.75
TESTING:
Accuracy of the network on the 10000 test images: 84.75 %
Average loss on the 10000 test images: 0.412
[34,   100] loss: 0.290 acc: 88.89 time: 9.34
[34,   200] loss: 0.286 acc: 88.62 time: 7.76
[34,   300] loss: 0.296 acc: 88.49 time: 9.20
TESTING:
Accuracy of the network on the 10000 test images: 84.92 %
Average loss on the 10000 test images: 0.408
[35,   100] loss: 0.285 acc: 89.13 time: 7.84
[35,   200] loss: 0.288 acc: 88.83 time: 8.85
[35,   300] loss: 0.293 acc: 88.59 time: 8.44
TESTING:
Accuracy of the network on the 10000 test images: 84.72 %
Average loss on the 10000 test images: 0.409
[36,   100] loss: 0.287 acc: 88.71 time: 9.24
[36,   200] loss: 0.283 acc: 89.09 time: 8.88
[36,   300] loss: 0.280 acc: 89.20 time: 7.84
TESTING:
Accuracy of the network on the 10000 test images: 84.66 %
Average loss on the 10000 test images: 0.415
[37,   100] loss: 0.281 acc: 89.10 time: 9.13
[37,   200] loss: 0.285 acc: 89.06 time: 7.67
[37,   300] loss: 0.293 acc: 88.40 time: 8.98
TESTING:
Accuracy of the network on the 10000 test images: 84.55 %
Average loss on the 10000 test images: 0.417
[38,   100] loss: 0.299 acc: 88.32 time: 7.87
[38,   200] loss: 0.286 acc: 88.91 time: 8.96
[38,   300] loss: 0.285 acc: 89.08 time: 7.65
TESTING:
Accuracy of the network on the 10000 test images: 84.68 %
Average loss on the 10000 test images: 0.411
[39,   100] loss: 0.293 acc: 88.59 time: 9.16
[39,   200] loss: 0.289 acc: 88.66 time: 7.80
[39,   300] loss: 0.291 acc: 88.63 time: 8.59
TESTING:
Accuracy of the network on the 10000 test images: 84.55 %
Average loss on the 10000 test images: 0.419
[40,   100] loss: 0.286 acc: 89.02 time: 7.53
[40,   200] loss: 0.292 acc: 88.62 time: 8.54
[40,   300] loss: 0.292 acc: 88.66 time: 8.47
TESTING:
Accuracy of the network on the 10000 test images: 84.34 %
Average loss on the 10000 test images: 0.414
[41,   100] loss: 0.288 acc: 88.86 time: 8.91
[41,   200] loss: 0.279 acc: 89.29 time: 8.34
```

```
[41,   300] loss: 0.293 acc: 88.77 time: 7.71
TESTING:
Accuracy of the network on the 10000 test images: 84.70 %
Average loss on the 10000 test images: 0.416
[42,   100] loss: 0.295 acc: 88.51 time: 8.00
[42,   200] loss: 0.289 acc: 88.76 time: 8.35
[42,   300] loss: 0.295 acc: 88.45 time: 8.57
TESTING:
Accuracy of the network on the 10000 test images: 84.64 %
Average loss on the 10000 test images: 0.413
[43,   100] loss: 0.289 acc: 88.77 time: 8.98
[43,   200] loss: 0.284 acc: 88.85 time: 8.72
[43,   300] loss: 0.294 acc: 88.44 time: 7.33
TESTING:
Accuracy of the network on the 10000 test images: 84.84 %
Average loss on the 10000 test images: 0.416
[44,   100] loss: 0.298 acc: 88.63 time: 8.82
[44,   200] loss: 0.286 acc: 89.03 time: 7.50
[44,   300] loss: 0.285 acc: 89.20 time: 8.63
TESTING:
Accuracy of the network on the 10000 test images: 84.74 %
Average loss on the 10000 test images: 0.416
[45,   100] loss: 0.278 acc: 89.03 time: 7.74
[45,   200] loss: 0.287 acc: 88.59 time: 8.62
[45,   300] loss: 0.296 acc: 88.48 time: 7.32
TESTING:
Accuracy of the network on the 10000 test images: 84.89 %
Average loss on the 10000 test images: 0.410
Finished Training
```

[ ]:
```
torch.save(resnet50.state_dict(), 'rotation_model_extra.pth')
```

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

from torchvision import models

# Load the previously trained ResNet18 model
net = models.resnet50(num_classes=4)

saved_path = 'rotation_model_extra.pth'
net.load_state_dict(torch.load(saved_path))

# Move the model to the device (GPU or CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
num_classes = 10
```

```
net.fc = nn.Linear(net.fc.in_features, num_classes).to(device)
```

[ ]:
```
for param in net.parameters():
    param.requires_grad = False

# Unfreeze the 'layer4' block and 'fc' layer
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.conv3.weight
         layer4.0.bn3.weight
         layer4.0.bn3.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
         layer4.1.conv3.weight
         layer4.1.bn3.weight
         layer4.1.bn3.bias
         layer4.2.conv1.weight
         layer4.2.bn1.weight
         layer4.2.bn1.bias
         layer4.2.conv2.weight
```

```
layer4.2.bn2.weight
layer4.2.bn2.bias
layer4.2.conv3.weight
layer4.2.bn3.weight
layer4.2.bn3.bias
fc.weight
fc.bias
```

[ ]:
```
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-3)
```

[ ]:
```
criterion = nn.CrossEntropyLoss()

net = net.to(device)
```

[ ]:
```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=4, init_lr=0.1,␣
 ↪task='classification')
```

```
[1,   100] loss: 1.629 acc: 37.02 time: 8.02
[1,   200] loss: 1.374 acc: 48.36 time: 6.24
[1,   300] loss: 1.333 acc: 50.48 time: 7.80
TESTING:
Accuracy of the network on the 10000 test images: 51.48 %
Average loss on the 10000 test images: 1.289
[2,   100] loss: 1.252 acc: 54.67 time: 6.54
[2,   200] loss: 1.245 acc: 54.31 time: 7.62
[2,   300] loss: 1.221 acc: 55.62 time: 6.44
TESTING:
Accuracy of the network on the 10000 test images: 58.33 %
Average loss on the 10000 test images: 1.151
[3,   100] loss: 1.208 acc: 56.43 time: 6.48
[3,   200] loss: 1.194 acc: 56.59 time: 8.67
[3,   300] loss: 1.191 acc: 56.94 time: 6.22
TESTING:
Accuracy of the network on the 10000 test images: 60.45 %
Average loss on the 10000 test images: 1.092
[4,   100] loss: 1.184 acc: 56.80 time: 8.16
[4,   200] loss: 1.186 acc: 57.83 time: 6.54
[4,   300] loss: 1.183 acc: 57.73 time: 7.93
TESTING:
Accuracy of the network on the 10000 test images: 55.69 %
Average loss on the 10000 test images: 1.211
[5,   100] loss: 1.107 acc: 60.01 time: 8.00
[5,   200] loss: 1.069 acc: 61.82 time: 6.80
[5,   300] loss: 1.057 acc: 62.01 time: 7.57
TESTING:
Accuracy of the network on the 10000 test images: 64.44 %
Average loss on the 10000 test images: 0.985
[6,   100] loss: 1.031 acc: 63.41 time: 6.65
```

```
[6,   200] loss: 1.025 acc: 62.99 time: 7.96
[6,   300] loss: 1.037 acc: 62.29 time: 6.34
TESTING:
Accuracy of the network on the 10000 test images: 65.02 %
Average loss on the 10000 test images: 0.964
[7,   100] loss: 1.015 acc: 63.13 time: 7.93
[7,   200] loss: 1.002 acc: 63.82 time: 6.49
[7,   300] loss: 1.020 acc: 63.36 time: 7.80
TESTING:
Accuracy of the network on the 10000 test images: 66.21 %
Average loss on the 10000 test images: 0.940
[8,   100] loss: 1.006 acc: 63.51 time: 8.22
[8,   200] loss: 0.999 acc: 64.25 time: 6.29
[8,   300] loss: 1.007 acc: 63.90 time: 7.88
TESTING:
Accuracy of the network on the 10000 test images: 66.80 %
Average loss on the 10000 test images: 0.930
[9,   100] loss: 0.987 acc: 63.98 time: 6.52
[9,   200] loss: 0.979 acc: 64.72 time: 7.85
[9,   300] loss: 0.966 acc: 65.34 time: 6.51
TESTING:
Accuracy of the network on the 10000 test images: 67.44 %
Average loss on the 10000 test images: 0.914
[10,   100] loss: 0.981 acc: 65.38 time: 7.26
[10,   200] loss: 0.960 acc: 65.77 time: 7.31
[10,   300] loss: 0.950 acc: 66.50 time: 6.97
TESTING:
Accuracy of the network on the 10000 test images: 67.36 %
Average loss on the 10000 test images: 0.909
[11,   100] loss: 0.969 acc: 65.66 time: 8.28
[11,   200] loss: 0.951 acc: 66.03 time: 6.41
[11,   300] loss: 0.945 acc: 65.71 time: 8.05
TESTING:
Accuracy of the network on the 10000 test images: 67.59 %
Average loss on the 10000 test images: 0.906
[12,   100] loss: 0.952 acc: 66.10 time: 6.85
[12,   200] loss: 0.955 acc: 65.66 time: 7.60
[12,   300] loss: 0.942 acc: 66.28 time: 6.55
TESTING:
Accuracy of the network on the 10000 test images: 67.62 %
Average loss on the 10000 test images: 0.900
[13,   100] loss: 0.930 acc: 66.48 time: 6.60
[13,   200] loss: 0.943 acc: 65.78 time: 8.21
[13,   300] loss: 0.952 acc: 66.20 time: 6.42
TESTING:
Accuracy of the network on the 10000 test images: 67.90 %
Average loss on the 10000 test images: 0.901
[14,   100] loss: 0.957 acc: 65.37 time: 8.27
```

```
[14,   200] loss: 0.929 acc: 66.56 time: 6.45
[14,   300] loss: 0.944 acc: 66.20 time: 8.20
TESTING:
Accuracy of the network on the 10000 test images: 68.13 %
Average loss on the 10000 test images: 0.896
[15,   100] loss: 0.943 acc: 66.59 time: 7.58
[15,   200] loss: 0.932 acc: 66.29 time: 7.34
[15,   300] loss: 0.942 acc: 66.51 time: 7.21
TESTING:
Accuracy of the network on the 10000 test images: 67.83 %
Average loss on the 10000 test images: 0.901
[16,   100] loss: 0.942 acc: 66.34 time: 6.57
[16,   200] loss: 0.937 acc: 66.05 time: 7.97
[16,   300] loss: 0.954 acc: 65.44 time: 6.43
TESTING:
Accuracy of the network on the 10000 test images: 67.90 %
Average loss on the 10000 test images: 0.899
[17,   100] loss: 0.945 acc: 66.09 time: 8.21
[17,   200] loss: 0.948 acc: 66.08 time: 6.39
[17,   300] loss: 0.943 acc: 65.88 time: 8.05
TESTING:
Accuracy of the network on the 10000 test images: 67.86 %
Average loss on the 10000 test images: 0.900
[18,   100] loss: 0.940 acc: 66.13 time: 8.05
[18,   200] loss: 0.937 acc: 66.30 time: 6.23
[18,   300] loss: 0.940 acc: 66.59 time: 8.11
TESTING:
Accuracy of the network on the 10000 test images: 67.85 %
Average loss on the 10000 test images: 0.900
[19,   100] loss: 0.945 acc: 66.30 time: 6.54
[19,   200] loss: 0.928 acc: 66.59 time: 7.99
[19,   300] loss: 0.944 acc: 66.06 time: 6.29
TESTING:
Accuracy of the network on the 10000 test images: 68.02 %
Average loss on the 10000 test images: 0.896
[20,   100] loss: 0.944 acc: 66.21 time: 7.40
[20,   200] loss: 0.949 acc: 65.70 time: 7.39
[20,   300] loss: 0.924 acc: 66.78 time: 7.69
TESTING:
Accuracy of the network on the 10000 test images: 68.07 %
Average loss on the 10000 test images: 0.898
Finished Training
```

Extra credit part 1.c

[3]:
```
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
    elif rot == 1: # 90 degrees rotation
        return torch.rot90(img.permute(1,2,0), 3).permute(2,0,1)
    elif rot == 2: # 180 degrees rotation
        return torch.rot90(img, 2)
    elif rot == 3: # 270 degrees rotation
        return torch.rot90(img.permute(1,2,0), 1).permute(2,0,1)

    # TODO: Implement rotate_img() - return the rotated img
    #
    #
    #
    else:
        raise ValueError('rotation should be 0, 90, 180, or 270 degrees')


class CIFAR10Rotation(torchvision.datasets.ImageFolder):

    def __init__(self, root, transform) -> None:
        super().__init__(root=root, transform=transform)

    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()
```

[4]:
```
from fastai.vision.all import *
path = untar_data(URLs.IMAGENETTE)
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

[5]:
```
path
```

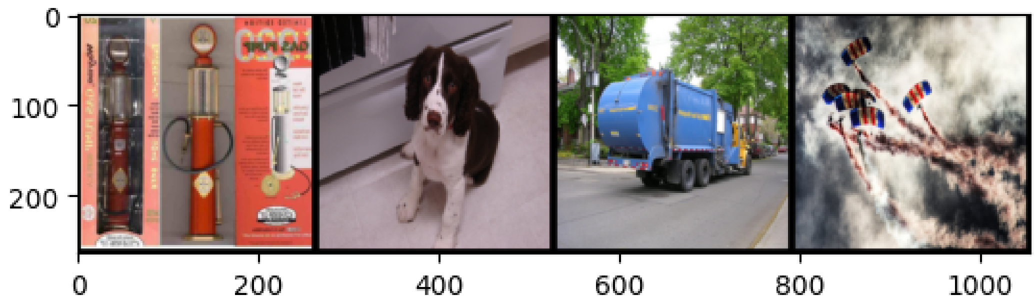
[5]: Path('/root/.fastai/data/imagenette2')

```
[6]: transform_train = transforms.Compose([
         transforms.Resize((256,256)),
         transforms.Pad(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=str(path)+'/train', transform=transform_train)
     trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                               shuffle=True, num_workers=2)

     testset = CIFAR10Rotation(root=str(path)+'/val', transform=transform_test)
     testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              shuffle=False, num_workers=2)
```

```
[7]: 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))
```

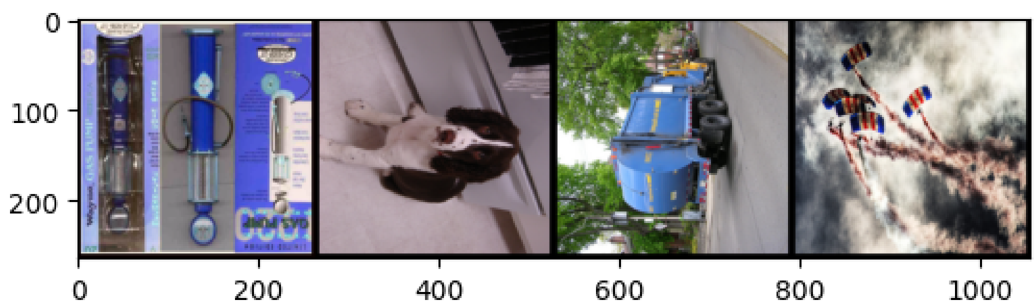
```python
print('Rotation labels: ', ' '.join(f'{rot_classes[rot_labels[j]]:5s}' for j in␣
↪range(4)))
```

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



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

Class labels: horse car   frog truck



Rotation labels: 180   90   270   0

[8]:
```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
            # The class with the highest energy is what we choose as prediction
            #
            #
            #
            # Calculate outputs by running images through the network


            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)

            #correct += labels.size(0)
            total += labels.size(0)
            # loss
            correct += (predicted == labels).sum().item()
            avg_test_loss += criterion(outputs, labels)  / len(testloader)

            # doubt on below line
            #total += avg_test_loss


    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}')
```

[9]:
```
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
```

[10]:
```
#device = 'cuda' if torch.cuda.is_available() else 'cpu'


device = 'cuda' if torch.cuda.is_available() else 'cpu'

device
```

[10]: 'cpu'

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

```python
from torchvision.models import resnet18

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

[11]:
```python
net = models.resnet50(pretrained=True)
num_features = net.fc.in_features
net.fc = nn.Linear(num_features, 10)
net = net.to(device)
```

```
/usr/local/lib/python3.9/dist-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(
/usr/local/lib/python3.9/dist-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=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|      | 97.8M/97.8M [00:01<00:00, 92.6MB/s]
```

[12]:
```python
device
```

[12]: 'cpu'

[13]:
```python
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
```

[14]:
```python
# 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()
```

```
    adjust_learning_rate(optimizer, epoch, init_lr, decay_epochs)


    for i, (imgs, imgs_rotated, rotation_label, cls_label) in␣
↪enumerate(trainloader, 0):


        # TODO: Set the data to the correct device; Different task will use␣
↪different inputs and labels
        #

        # TODO: Zero the parameter gradients
        #

        # TODO: forward + backward + optimize
        #
        #
        #
        #
        #
        # Set the data to the correct device; Different task will use␣
↪different inputs and labels
        if task == 'rotation':
          images, labels = imgs_rotated.to(device), rotation_label.
↪to(device)
        elif task == 'classification':
          images, labels = imgs.to(device), cls_label.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # TODO: Get predicted results

        predicted = torch.argmax(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')
```

[15]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net.to(device)
```

[15]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
```

```
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
```

```
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=10, bias=True)
)
```

[ ]:
```
train(net, criterion, optimizer, num_epochs=4, decay_epochs=5, init_lr=0.01,␣
↪task='classification')
```

We have trained both

[6]:
```
import os
import numpy as np
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets import ImageFolder
import torchvision.models as models
```

```python
from PIL import Image

# 1. Download the ImageNette dataset
# Go to https://github.com/fastai/imagenette and follow the instructions to␣
↪download the dataset

# 2. Create a custom dataset loader
class ImageNetteRotDataset(Dataset):
    def __init__(self, root, transform=None):
        self.data = ImageFolder(root)
        self.transform = transform

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

    def __getitem__(self, index):
        img, _ = self.data[index]
        rot_label = np.random.choice([0, 1, 2, 3])
        img = img.rotate(90 * rot_label)

        if self.transform:
            img = self.transform(img)

        return img, rot_label

# 3. Prepare data transformations
data_transforms = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Set the dataset path
imagenette_train_path = '/content/drive/MyDrive/assignment3_starter/
↪assignment3_part1/imagenette2-160/train'

# 4. Instantiate the model
resnet50 = models.resnet50(pretrained=True)
num_features = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_features, 4)

# 5. Train the model on the rotation prediction task
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
resnet50 = resnet50.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(resnet50.parameters(), lr=0.001, momentum=0.9)
```

```python
num_epochs = 10
train_dataset = ImageNetteRotDataset(imagenette_train_path,
    transform=data_transforms)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
    num_workers=4)

for epoch in range(num_epochs):
    running_loss = 0.0
    correct = 0
    total = 0

    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = resnet50(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / (i + 1)
    epoch_acc = correct / total * 100
    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {epoch_loss:.4f}, Accuracy:
    {epoch_acc:.2f}%')

print('Finished training the rotation prediction model.')
```

```
Epoch 1/10, Loss: 0.8177, Accuracy: 62.93%
Epoch 2/10, Loss: 0.3756, Accuracy: 85.67%
Epoch 3/10, Loss: 0.2964, Accuracy: 89.21%
Epoch 4/10, Loss: 0.2598, Accuracy: 90.52%
Epoch 5/10, Loss: 0.2168, Accuracy: 91.95%
Epoch 6/10, Loss: 0.1750, Accuracy: 93.64%
Epoch 7/10, Loss: 0.1587, Accuracy: 94.43%
Epoch 8/10, Loss: 0.1389, Accuracy: 95.28%
Epoch 9/10, Loss: 0.1303, Accuracy: 95.63%
Epoch 10/10, Loss: 0.1152, Accuracy: 95.91%
Finished training the rotation prediction model.
```