(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.

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

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

```
# fill in the path in your Google Drive in the string below. Note: do not escape slashes or spaces
import os
datadir = "/content/assignment3"
if not os.path.exists(datadir):
  !ln -s "/content/drive/MyDrive/assignment3_starter" $datadir # TODO: Fill your A3 path
  datadir = "/content/drive/MyDrive/assignment3_starter"
os.chdir(datadir)
!pwd
```

```
/content/drive/.shortcut-targets-by-id/1C6vzFdqC8Hjc1SzmCGHlAtkpvu5roImn/assignment3_starter
```

Data Setup

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

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

```
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np
import random


def rotate_img(img, rot):
    if rot == 0: # 0 degrees rotation
        return img
    # TODO: Implement rotate_img() - return the rotated img
    elif rot == 1 or 2 or 3: # 90, 180 or 270 degrees rotation counter-clockwise
        return transforms.functional.rotate(img=img, angle=(rot*90))
    # End TODO
    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(),
```

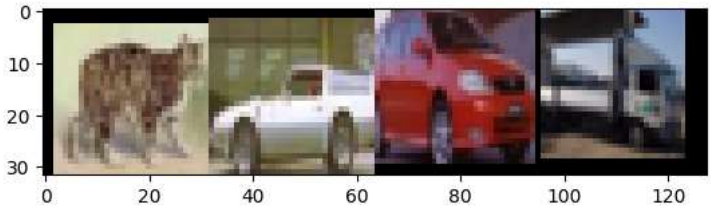
```
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

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
```

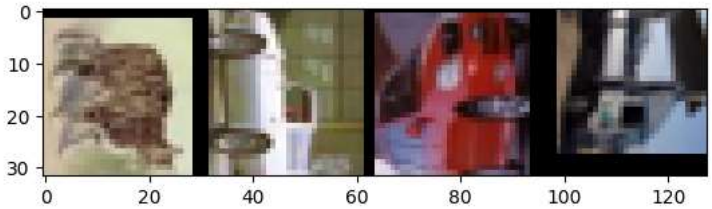
Show some example images and rotated images with labels:

```
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 RG
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RG
Class labels:  cat   car   car   truck
```




```
Rotation labels:  270   270   90    270
```

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
            outputs = net(images)
            predictions = outputs.squeeze(0).softmax(1).argmax(dim=1)
            batch_correct = torch.sum(predictions == labels)
            correct += batch_correct
            total += predictions.shape[0]
            # End TODO

            # loss
            avg_test_loss += criterion(outputs, labels)  / len(testloader)

    print('TESTING:')
    print(f'Accuracy of the network on the 10000 test images: {100 * correct / total:.2f} %')
    print(f'Average loss on the 10000 test images: {avg_test_loss:.3f}')


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 resnet50 on the rotation task

In this section, we will train a resnet50 model on the rotation task. The input is a rotated image and the model predicts the rotation label. See the Data Setup section for details.

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

```
'cuda'
```

```
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet50

net = resnet50(num_classes=4)
net = net.to(device)
```

```
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr = 0.01)
# Which to choose?? Are we free to choose what we like?

# TODO: Define criterion and optimizer



# Both the self-supervised rotation task and supervised CIFAR10 classification are
# trained with the CrossEntropyLoss, so we can use the training loop code.

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

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

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

        net.train()

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

            # TODO: Set the data to the correct device; Different task will use different inputs and labels
            if task == 'rotation':
                images, labels = imgs_rotated.to(device), rotation_label.to(device)
            elif task == 'classification':
                images, labels = imgs.to(device), cls_label.to(device)
            else:
                raise ValueError('Task should either be classification or rotation')

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

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

            # TODO: Get predicted results
            predicted = outputs.squeeze(0).softmax(1).argmax(dim=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:
                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)
        # ENd of TODO

    print('Finished Training')


train(net, criterion, optimizer, num_epochs=45, decay_epochs=10, init_lr=0.01, task='rotation')

# TODO: Save the model
torch.save(net.state_dict(), '/content/drive/MyDrive/assignment3_starter/resnet50_weights')
```

```
[35,   200] loss: 0.690 acc: 72.05 time: 10.28
[35,   300] loss: 0.689 acc: 72.38 time: 11.32
TESTING:
Accuracy of the network on the 10000 test images: 72.75 %
Average loss on the 10000 test images: 0.676
[36,   100] loss: 0.714 acc: 71.31 time: 11.85
[36,   200] loss: 0.678 acc: 73.37 time: 11.37
[36,   300] loss: 0.686 acc: 72.41 time: 10.37
TESTING:
Accuracy of the network on the 10000 test images: 73.27 %
Average loss on the 10000 test images: 0.677
[37,   100] loss: 0.694 acc: 71.88 time: 10.35
[37,   200] loss: 0.687 acc: 72.54 time: 11.64
[37,   300] loss: 0.699 acc: 71.98 time: 11.33
TESTING:
Accuracy of the network on the 10000 test images: 73.18 %
Average loss on the 10000 test images: 0.679
[38,   100] loss: 0.691 acc: 71.59 time: 11.49
[38,   200] loss: 0.684 acc: 72.73 time: 9.82
[38,   300] loss: 0.694 acc: 72.23 time: 11.65
TESTING:
Accuracy of the network on the 10000 test images: 73.08 %
Average loss on the 10000 test images: 0.677
[39,   100] loss: 0.693 acc: 71.84 time: 11.78
[39,   200] loss: 0.685 acc: 72.34 time: 11.58
[39,   300] loss: 0.691 acc: 72.30 time: 10.79
TESTING:
Accuracy of the network on the 10000 test images: 72.69 %
Average loss on the 10000 test images: 0.682
[40,   100] loss: 0.699 acc: 71.88 time: 10.53
[40,   200] loss: 0.688 acc: 72.23 time: 11.60
[40,   300] loss: 0.679 acc: 72.61 time: 11.57
TESTING:
Accuracy of the network on the 10000 test images: 73.18 %
Average loss on the 10000 test images: 0.679
[41,   100] loss: 0.691 acc: 72.22 time: 11.79
[41,   200] loss: 0.695 acc: 71.80 time: 10.36
```

Fine-tuning on the pre-trained model

In this section, we will load the pre-trained resnet50 model and fine-tune on the classification task. We will freeze all previous layers except for the 'layer4' block and 'fc' layer.

```
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet50

# TODO: Load the pre-trained resnet50 model
saved_state_dict = torch.load('/content/drive/MyDrive/assignment3_starter/resnet50_weights')
loaded_net = resnet50(num_classes=4)
loaded_net.load_state_dict(saved_state_dict)
loaded_net = loaded_net.to(device)
criterion = nn.CrossEntropyLoss()
run_test(loaded_net, testloader, criterion, 'rotation') # Verification, should give ~78% accuracy
# we also need to change no of outputs for the image label classification task
# to do this, reference : https://discuss.pytorch.org/t/how-to-reshape-last-layer-of-pytorch-cnn-model-while-doing-transfer-learning/62681/2
old_input_features = loaded_net.fc.in_features
loaded_net.fc = nn.Linear(old_input_features, 10)
loaded_net = loaded_net.to(device)  # this is necessary after replacing the fc layer
```

```
TESTING:
Accuracy of the network on the 10000 test images: 74.88 %
Average loss on the 10000 test images: 0.637
```

```
#run_test(net, testloader, criterion, 'rotation')
run_test(loaded_net, testloader, criterion, 'classification')  # should be close to random (10%) since model hasn't been trained on CIFAR-10
#run_test(net, testloader, criterion, 'classification')
```

```
TESTING:
Accuracy of the network on the 10000 test images: 8.31 %
Average loss on the 10000 test images: 2.303
```

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

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


print(type(loaded_net))
print(type(loaded_net.parameters))
print(type(loaded_net.layer4))
print(type(loaded_net.fc))
# print(type(loaded_net.features))
print(type(loaded_net.layer4.parameters))
print(type(loaded_net.fc.parameters))
# print(type(loaded_net.features.parameters))
#print(loaded_net)
```

```
<class 'torchvision.models.resnet.ResNet'>
<class 'method'>
<class 'torch.nn.modules.container.Sequential'>
<class 'torch.nn.modules.linear.Linear'>
<class 'method'>
<class 'method'>
```

```
# Print all the trainable parameters (they should be parameters of the last 2 layers)
params_to_update = loaded_net.parameters()
print("Params to learn:")
params_to_update = []
for name,param in loaded_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
```

```
# TODO: Define criterion and optimizer
# Note that your optimizer only needs to update the parameters that are trainable.
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params_to_update, lr = 0.01)


train(loaded_net, criterion, optimizer, num_epochs=20, decay_epochs=9, init_lr=0.01, task='classification')
torch.save(loaded_net.state_dict(), '/content/drive/MyDrive/assignment3_starter/pretrained_resnet50_last_two_layers_fine_tuned_weights')
```

```
[1,   100] loss: 1.971 acc: 25.88 time: 9.50
[1,   200] loss: 1.709 acc: 36.65 time: 9.19
[1,   300] loss: 1.639 acc: 39.28 time: 7.96
TESTING:
Accuracy of the network on the 10000 test images: 43.03 %
Average loss on the 10000 test images: 1.602
```

```
[2,   100] loss: 1.561 acc: 42.95 time: 9.70
[2,   200] loss: 1.567 acc: 42.82 time: 8.16
[2,   300] loss: 1.516 acc: 45.43 time: 9.30
TESTING:
Accuracy of the network on the 10000 test images: 45.06 %
Average loss on the 10000 test images: 1.521
[3,   100] loss: 1.507 acc: 45.15 time: 9.66
[3,   200] loss: 1.488 acc: 45.49 time: 7.75
[3,   300] loss: 1.493 acc: 45.65 time: 9.67
TESTING:
Accuracy of the network on the 10000 test images: 47.28 %
Average loss on the 10000 test images: 1.477
[4,   100] loss: 1.467 acc: 47.14 time: 8.30
[4,   200] loss: 1.453 acc: 46.88 time: 9.37
[4,   300] loss: 1.451 acc: 46.80 time: 9.64
TESTING:
Accuracy of the network on the 10000 test images: 46.28 %
Average loss on the 10000 test images: 1.489
[5,   100] loss: 1.436 acc: 47.44 time: 7.99
[5,   200] loss: 1.435 acc: 47.69 time: 9.51
[5,   300] loss: 1.429 acc: 48.22 time: 8.34
TESTING:
Accuracy of the network on the 10000 test images: 44.99 %
Average loss on the 10000 test images: 1.534
[6,   100] loss: 1.410 acc: 49.08 time: 9.77
[6,   200] loss: 1.413 acc: 48.58 time: 9.57
[6,   300] loss: 1.410 acc: 49.27 time: 7.93
TESTING:
Accuracy of the network on the 10000 test images: 47.56 %
Average loss on the 10000 test images: 1.468
[7,   100] loss: 1.397 acc: 49.57 time: 9.78
[7,   200] loss: 1.395 acc: 48.91 time: 8.42
[7,   300] loss: 1.395 acc: 49.68 time: 9.05
TESTING:
Accuracy of the network on the 10000 test images: 49.72 %
Average loss on the 10000 test images: 1.405
[8,   100] loss: 1.380 acc: 49.73 time: 9.93
[8,   200] loss: 1.383 acc: 50.15 time: 7.90
[8,   300] loss: 1.368 acc: 50.16 time: 9.67
TESTING:
Accuracy of the network on the 10000 test images: 51.57 %
Average loss on the 10000 test images: 1.369
[9,   100] loss: 1.368 acc: 50.17 time: 8.94
[9,   200] loss: 1.358 acc: 50.96 time: 8.54
[9,   300] loss: 1.358 acc: 50.75 time: 9.65
TESTING:
Accuracy of the network on the 10000 test images: 52.75 %
Average loss on the 10000 test images: 1.321
[10,   100] loss: 1.303 acc: 52.66 time: 8.01
[10,   200] loss: 1.291 acc: 53.27 time: 9.68
[10,   300] loss: 1.299 acc: 52.78 time: 9.39
TESTING:
```

Fine-tuning on the randomly initialized model

In this section, we will randomly initialize a resnet50 model and fine-tune on the classification task. We will freeze all previous layers except for the 'layer4' block and 'fc' layer.

```
import torch.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet50

# TODO: Randomly initialize a resnet50 model
r_init_net = resnet50(num_classes=10).to(device)


# TODO: Freeze all previous layers; only keep the 'layer4' block and 'fc' layer trainable
# To do this, you should set requires_grad=False for the frozen layers.
for param in r_init_net.parameters():
    param.requires_grad = False
for param in r_init_net.layer4.parameters():
    param.requires_grad = True
for param in r_init_net.fc.parameters():
    param.requires_grad = True


# Print all the trainable parameters
params_to_update = r_init_net.parameters()
print("Params to learn:")
```

```
params_to_update = []
for name,param in r_init_net.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)
        print("\t",name)


# TODO: Define criterion and optimizer
# Note that your optimizer only needs to update the parameters that are trainable.
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(params_to_update, lr = 0.01)


train(r_init_net, criterion, optimizer, num_epochs=20, decay_epochs=10, init_lr=0.01, task='classification')
torch.save(r_init_net.state_dict(), '/content/drive/MyDrive/assignment3_starter/random_init_resnet50_last_two_layers_fine_tuned_weights')
```

```
[1,   100] loss: 4.038 acc: 13.95 time: 8.17
[1,   200] loss: 2.681 acc: 17.08 time: 8.91
[1,   300] loss: 2.248 acc: 17.42 time: 9.06
TESTING:
Accuracy of the network on the 10000 test images: 20.22 %
Average loss on the 10000 test images: 2.136
[2,   100] loss: 2.162 acc: 19.41 time: 10.12
[2,   200] loss: 2.135 acc: 20.34 time: 9.24
[2,   300] loss: 2.138 acc: 20.57 time: 7.58
TESTING:
Accuracy of the network on the 10000 test images: 21.78 %
Average loss on the 10000 test images: 2.092
[3,   100] loss: 2.121 acc: 21.05 time: 9.49
[3,   200] loss: 2.116 acc: 21.16 time: 8.66
[3,   300] loss: 2.101 acc: 22.20 time: 8.03
TESTING:
Accuracy of the network on the 10000 test images: 23.19 %
Average loss on the 10000 test images: 2.055
[4,   100] loss: 2.096 acc: 21.82 time: 9.23
[4,   200] loss: 2.093 acc: 21.61 time: 7.54
[4,   300] loss: 2.088 acc: 22.38 time: 9.03
TESTING:
Accuracy of the network on the 10000 test images: 23.93 %
Average loss on the 10000 test images: 2.048
[5,   100] loss: 2.083 acc: 22.48 time: 7.64
[5,   200] loss: 2.077 acc: 22.85 time: 9.24
[5,   300] loss: 2.072 acc: 23.17 time: 8.09
TESTING:
Accuracy of the network on the 10000 test images: 25.43 %
Average loss on the 10000 test images: 2.017
[6,   100] loss: 2.061 acc: 23.54 time: 9.29
[6,   200] loss: 2.070 acc: 23.16 time: 8.85
[6,   300] loss: 2.069 acc: 23.79 time: 7.62
TESTING:
Accuracy of the network on the 10000 test images: 25.84 %
Average loss on the 10000 test images: 2.014
[7,   100] loss: 2.060 acc: 23.73 time: 9.47
[7,   200] loss: 2.066 acc: 23.38 time: 7.49
[7,   300] loss: 2.067 acc: 23.47 time: 9.25
TESTING:
Accuracy of the network on the 10000 test images: 26.45 %
Average loss on the 10000 test images: 2.001
[8,   100] loss: 2.056 acc: 23.71 time: 7.61
[8,   200] loss: 2.059 acc: 24.34 time: 9.27
[8,   300] loss: 2.053 acc: 24.09 time: 8.14
TESTING:
Accuracy of the network on the 10000 test images: 26.70 %
Average loss on the 10000 test images: 2.009
[9,   100] loss: 2.039 acc: 24.95 time: 9.51
[9,   200] loss: 2.046 acc: 24.52 time: 9.25
[9,   300] loss: 2.044 acc: 24.73 time: 7.51
TESTING:
Accuracy of the network on the 10000 test images: 26.85 %
Average loss on the 10000 test images: 1.985
[10,   100] loss: 2.033 acc: 25.44 time: 9.38
[10,   200] loss: 2.040 acc: 24.81 time: 7.44
[10,   300] loss: 2.042 acc: 24.92 time: 9.12
TESTING:
```

Supervised training on the pre-trained model

In this section, we will load the pre-trained resnet50 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 resnet50

# TODO: Load the pre-trained resnet50 model
saved_state_dict = torch.load('/content/drive/MyDrive/assignment3_starter/resnet50_weights')
loaded_net2 = resnet50(num_classes=4)
loaded_net2.load_state_dict(saved_state_dict)
loaded_net2 = loaded_net2.to(device)
run_test(loaded_net2, testloader, criterion, 'rotation')  # Verification, should give ~78% accuracy
# we also need to change no of outputs for the image label classification task
# to do this, reference : https://discuss.pytorch.org/t/how-to-reshape-last-layer-of-pytorch-cnn-model-while-doing-transfer-learning/62681/2
old_input_features = loaded_net2.fc.in_features
loaded_net2.fc = nn.Linear(old_input_features, 10)
loaded_net2 = loaded_net2.to(device) # this is necessary after replacing the fc layer


# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(loaded_net2.parameters(), lr = 0.01)


train(loaded_net2, criterion, optimizer, num_epochs=20, decay_epochs=9, init_lr=0.01, task='classification')
torch.save(loaded_net2.state_dict(), '/content/drive/MyDrive/assignment3_starter/pretrained_resnet50_all_layers_fine_tuned_weights')
```

```
TESTING:
Accuracy of the network on the 10000 test images: 82.58 %
Average loss on the 10000 test images: 0.510
[12,   100] loss: 0.436 acc: 84.89 time: 10.61
[12,   200] loss: 0.435 acc: 85.08 time: 10.50
[12,   300] loss: 0.442 acc: 84.61 time: 9.81
TESTING:
Accuracy of the network on the 10000 test images: 82.80 %
Average loss on the 10000 test images: 0.508
[13,   100] loss: 0.426 acc: 85.40 time: 9.75
[13,   200] loss: 0.439 acc: 84.88 time: 10.77
[13,   300] loss: 0.437 acc: 84.83 time: 10.64
TESTING:
Accuracy of the network on the 10000 test images: 82.81 %
Average loss on the 10000 test images: 0.504
[14,   100] loss: 0.440 acc: 84.66 time: 9.31
[14,   200] loss: 0.420 acc: 85.46 time: 10.73
[14,   300] loss: 0.441 acc: 84.53 time: 10.74
TESTING:
Accuracy of the network on the 10000 test images: 82.80 %
Average loss on the 10000 test images: 0.506
[15,   100] loss: 0.440 acc: 84.38 time: 10.60
[15,   200] loss: 0.432 acc: 84.98 time: 9.69
[15,   300] loss: 0.432 acc: 85.14 time: 10.97
TESTING:
Accuracy of the network on the 10000 test images: 82.90 %
Average loss on the 10000 test images: 0.505
[16,   100] loss: 0.427 acc: 85.09 time: 11.10
[16,   200] loss: 0.440 acc: 85.14 time: 9.56
[16,   300] loss: 0.441 acc: 84.39 time: 10.39
TESTING:
Accuracy of the network on the 10000 test images: 82.51 %
Average loss on the 10000 test images: 0.509
[17,   100] loss: 0.418 acc: 85.53 time: 11.06
[17,   200] loss: 0.437 acc: 84.84 time: 10.37
[17,   300] loss: 0.439 acc: 84.70 time: 9.48
TESTING:
Accuracy of the network on the 10000 test images: 82.69 %
Average loss on the 10000 test images: 0.505
[18,   100] loss: 0.432 acc: 85.05 time: 11.09
[18,   200] loss: 0.428 acc: 85.11 time: 10.66
[18,   300] loss: 0.433 acc: 85.06 time: 9.14
TESTING:
Accuracy of the network on the 10000 test images: 82.82 %
Average loss on the 10000 test images: 0.506
[19,   100] loss: 0.432 acc: 84.95 time: 10.81
[19,   200] loss: 0.432 acc: 85.12 time: 10.66
[19,   300] loss: 0.436 acc: 84.70 time: 10.56
TESTING:
Accuracy of the network on the 10000 test images: 82.60 %
Average loss on the 10000 test images: 0.508
[20,   100] loss: 0.431 acc: 84.45 time: 9.38
[20,   200] loss: 0.453 acc: 84.45 time: 10.66
[20,   300] loss: 0.428 acc: 85.16 time: 10.65
TESTING:
Accuracy of the network on the 10000 test images: 82.77 %
Average loss on the 10000 test images: 0.507
Finished Training
```

Supervised training on the randomly initialized model

In this section, we will randomly initialize a resnet50 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 resnet50

# TODO: Randomly initialize a resnet50 model
r_init_net2 = resnet50(num_classes = 10).to(device)


# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(r_init_net2.parameters(), lr = 0.01)


train(r_init_net2, criterion, optimizer, num_epochs=20, decay_epochs=9, init_lr=0.01, task='classification')
torch.save(r_init_net2.state_dict(), '/content/drive/MyDrive/assignment3_starter/random_init_resnet50_all_layers_fine_tuned_weights')
```

```
TESTING:
Accuracy of the network on the 10000 test images: 82.02 %
Average loss on the 10000 test images: 0.518
[12,   100] loss: 0.476 acc: 83.54 time: 10.16
[12,   200] loss: 0.480 acc: 83.16 time: 10.35
[12,   300] loss: 0.488 acc: 83.42 time: 9.62
TESTING:
Accuracy of the network on the 10000 test images: 82.37 %
Average loss on the 10000 test images: 0.514
[13,   100] loss: 0.480 acc: 83.30 time: 9.95
[13,   200] loss: 0.443 acc: 84.73 time: 10.37
[13,   300] loss: 0.473 acc: 83.72 time: 10.38
TESTING:
Accuracy of the network on the 10000 test images: 82.90 %
Average loss on the 10000 test images: 0.504
[14,   100] loss: 0.447 acc: 84.64 time: 9.07
[14,   200] loss: 0.459 acc: 83.93 time: 10.58
[14,   300] loss: 0.445 acc: 84.91 time: 10.33
TESTING:
Accuracy of the network on the 10000 test images: 82.53 %
Average loss on the 10000 test images: 0.500
[15,   100] loss: 0.437 acc: 84.70 time: 9.62
[15,   200] loss: 0.444 acc: 84.43 time: 9.96
[15,   300] loss: 0.453 acc: 84.25 time: 10.30
TESTING:
Accuracy of the network on the 10000 test images: 82.86 %
Average loss on the 10000 test images: 0.498
[16,   100] loss: 0.442 acc: 84.83 time: 9.98
[16,   200] loss: 0.433 acc: 85.19 time: 9.57
[16,   300] loss: 0.435 acc: 84.94 time: 10.39
TESTING:
Accuracy of the network on the 10000 test images: 82.79 %
Average loss on the 10000 test images: 0.497
[17,   100] loss: 0.450 acc: 84.48 time: 10.45
[17,   200] loss: 0.421 acc: 85.46 time: 8.98
[17,   300] loss: 0.422 acc: 85.34 time: 10.21
TESTING:
Accuracy of the network on the 10000 test images: 83.13 %
Average loss on the 10000 test images: 0.491
[18,   100] loss: 0.416 acc: 85.41 time: 10.69
[18,   200] loss: 0.414 acc: 85.37 time: 9.20
[18,   300] loss: 0.420 acc: 85.42 time: 10.69
TESTING:
Accuracy of the network on the 10000 test images: 83.37 %
Average loss on the 10000 test images: 0.485
[19,   100] loss: 0.406 acc: 86.12 time: 10.85
[19,   200] loss: 0.401 acc: 86.16 time: 9.94
[19,   300] loss: 0.401 acc: 85.86 time: 10.17
TESTING:
Accuracy of the network on the 10000 test images: 83.47 %
Average loss on the 10000 test images: 0.480
[20,   100] loss: 0.417 acc: 85.46 time: 10.85
[20,   200] loss: 0.397 acc: 86.03 time: 10.73
[20,   300] loss: 0.401 acc: 86.34 time: 9.03
TESTING:
Accuracy of the network on the 10000 test images: 83.56 %
Average loss on the 10000 test images: 0.479
Finished Training
```

✓ 37m 31s completed at 4:48 PM