(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")
```

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

```
# 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(),
```

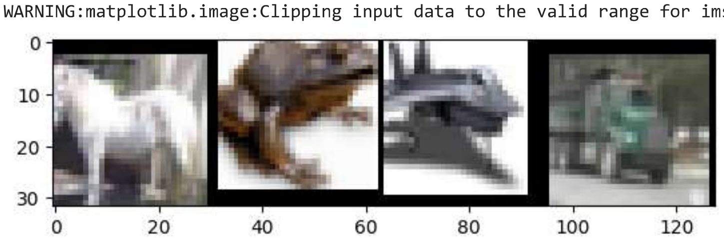
```
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

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
```

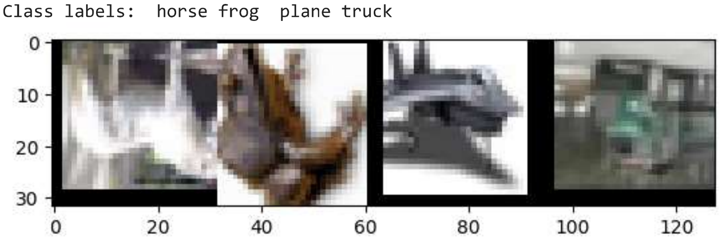
Show some example images and rotated images with labels:

```
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:  horse frog  plane truck
```




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

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 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'
```

```
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.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=15, init_lr=0.01, task='rotation')

# TODO: Save the model
torch.save(net.state_dict(), '/content/drive/MyDrive/assignment3_starter/resnet18_weights')
```

```
[40,   100] loss: 0.553 acc: 78.23 time: 9.66
[40,   200] loss: 0.570 acc: 77.95 time: 8.26
[40,   300] loss: 0.560 acc: 78.23 time: 8.86
TESTING:
Accuracy of the network on the 10000 test images: 78.16 %
Average loss on the 10000 test images: 0.557
[41,   100] loss: 0.571 acc: 77.80 time: 9.63
[41,   200] loss: 0.554 acc: 78.59 time: 7.69
[41,   300] loss: 0.564 acc: 77.75 time: 9.41
TESTING:
Accuracy of the network on the 10000 test images: 78.33 %
Average loss on the 10000 test images: 0.553
[42,   100] loss: 0.553 acc: 78.80 time: 8.01
[42,   200] loss: 0.565 acc: 78.10 time: 9.51
[42,   300] loss: 0.566 acc: 77.75 time: 9.41
TESTING:
Accuracy of the network on the 10000 test images: 78.57 %
Average loss on the 10000 test images: 0.554
[43,   100] loss: 0.562 acc: 77.96 time: 8.81
[43,   200] loss: 0.556 acc: 78.38 time: 9.35
[43,   300] loss: 0.545 acc: 78.46 time: 7.80
TESTING:
Accuracy of the network on the 10000 test images: 78.79 %
Average loss on the 10000 test images: 0.553
[44,   100] loss: 0.550 acc: 78.47 time: 9.64
[44,   200] loss: 0.550 acc: 78.43 time: 8.28
[44,   300] loss: 0.560 acc: 78.21 time: 8.97
TESTING:
Accuracy of the network on the 10000 test images: 77.79 %
Average loss on the 10000 test images: 0.559
[45,   100] loss: 0.568 acc: 78.23 time: 9.55
[45,   200] loss: 0.547 acc: 78.45 time: 8.05
[45,   300] loss: 0.560 acc: 78.48 time: 9.58
TESTING:
Accuracy of the network on the 10000 test images: 78.72 %
Average loss on the 10000 test images: 0.547
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.nn as nn
import torch.nn.functional as F

from torchvision.models import resnet18

# TODO: Load the pre-trained ResNet18 model
saved_state_dict = torch.load('/content/drive/MyDrive/assignment3_starter/resnet18_weights')
loaded_net = resnet18(num_classes=4)
loaded_net.load_state_dict(saved_state_dict)
loaded_net = loaded_net.to(device)
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: 77.00 %
Average loss on the 10000 test images: 0.596
```

```
#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: 10.77 %
Average loss on the 10000 test images: 2.311
```

```
# 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.downsample.0.weight
         layer4.0.downsample.1.weight
         layer4.0.downsample.1.bias
         layer4.1.conv1.weight
         layer4.1.bn1.weight
         layer4.1.bn1.bias
         layer4.1.conv2.weight
         layer4.1.bn2.weight
         layer4.1.bn2.bias
         fc.weight
         fc.bias
```

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


train(loaded_net, criterion, optimizer, num_epochs=20, decay_epochs=10, init_lr=0.01, task='classification')
torch.save(loaded_net.state_dict(), '/content/drive/MyDrive/assignment3_starter/pretrained_resnet18_last_two_layers_fine_tuned_weights')
```

```
[15,   200] loss: 1.186 acc: 57.29 time: 7.25
[15,   300] loss: 1.182 acc: 57.37 time: 8.84
TESTING:
Accuracy of the network on the 10000 test images: 58.75 %
Average loss on the 10000 test images: 1.164
[16,   100] loss: 1.170 acc: 57.84 time: 7.16
[16,   200] loss: 1.176 acc: 57.41 time: 8.80
[16,   300] loss: 1.185 acc: 57.32 time: 7.15
TESTING:
Accuracy of the network on the 10000 test images: 58.61 %
Average loss on the 10000 test images: 1.152
[17,   100] loss: 1.159 acc: 58.30 time: 8.96
[17,   200] loss: 1.184 acc: 57.97 time: 7.18
[17,   300] loss: 1.189 acc: 57.67 time: 8.78
TESTING:
Accuracy of the network on the 10000 test images: 58.48 %
Average loss on the 10000 test images: 1.162
[18,   100] loss: 1.187 acc: 57.26 time: 7.39
[18,   200] loss: 1.167 acc: 58.25 time: 8.90
[18,   300] loss: 1.170 acc: 57.60 time: 8.46
TESTING:
Accuracy of the network on the 10000 test images: 58.82 %
Average loss on the 10000 test images: 1.152
[19,   100] loss: 1.169 acc: 58.20 time: 7.36
[19,   200] loss: 1.173 acc: 57.62 time: 8.93
[19,   300] loss: 1.174 acc: 57.35 time: 7.35
TESTING:
Accuracy of the network on the 10000 test images: 58.95 %
Average loss on the 10000 test images: 1.145
[20,   100] loss: 1.180 acc: 57.27 time: 9.14
[20,   200] loss: 1.167 acc: 57.74 time: 7.32
[20,   300] loss: 1.164 acc: 58.17 time: 9.05
TESTING:
Accuracy of the network on the 10000 test images: 59.01 %
Average loss on the 10000 test images: 1.144
Finished Training
```

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.

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

from torchvision.models import resnet18

# TODO: Randomly initialize a ResNet18 model
r_init_net = resnet18(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)
```

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

```
            layer4.1.conv1.weight
            layer4.1.bn1.weight
            layer4.1.bn1.bias
            layer4.1.conv2.weight
            layer4.1.bn2.weight
            layer4.1.bn2.bias
            fc.weight
            fc.bias
```

```
# TODO: Define criterion and optimizer
# Note that your optimizer only needs to update the parameters that are trainable.
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(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_resnet18_last_two_layers_fine_tuned_weights')
```

```
TESTING:
Accuracy of the network on the 10000 test images: 44.48 %
Average loss on the 10000 test images: 1.541
[12,   100] loss: 1.588 acc: 42.78 time: 7.76
[12,   200] loss: 1.599 acc: 42.29 time: 8.70
[12,   300] loss: 1.585 acc: 43.07 time: 6.99
TESTING:
Accuracy of the network on the 10000 test images: 45.14 %
Average loss on the 10000 test images: 1.529
[13,   100] loss: 1.595 acc: 42.29 time: 8.84
[13,   200] loss: 1.591 acc: 42.73 time: 6.89
[13,   300] loss: 1.576 acc: 43.34 time: 8.79
TESTING:
Accuracy of the network on the 10000 test images: 45.17 %
Average loss on the 10000 test images: 1.524
[14,   100] loss: 1.568 acc: 43.43 time: 7.29
[14,   200] loss: 1.567 acc: 43.94 time: 8.68
[14,   300] loss: 1.583 acc: 43.01 time: 7.09
TESTING:
Accuracy of the network on the 10000 test images: 45.23 %
Average loss on the 10000 test images: 1.520
[15,   100] loss: 1.580 acc: 43.59 time: 8.85
[15,   200] loss: 1.557 acc: 44.09 time: 6.97
[15,   300] loss: 1.567 acc: 43.80 time: 8.60
TESTING:
Accuracy of the network on the 10000 test images: 45.30 %
Average loss on the 10000 test images: 1.519
[16,   100] loss: 1.560 acc: 44.08 time: 7.26
[16,   200] loss: 1.577 acc: 43.04 time: 8.70
[16,   300] loss: 1.558 acc: 43.92 time: 7.00
TESTING:
Accuracy of the network on the 10000 test images: 45.23 %
Average loss on the 10000 test images: 1.514
[17,   100] loss: 1.577 acc: 43.82 time: 8.99
[17,   200] loss: 1.575 acc: 43.93 time: 6.99
[17,   300] loss: 1.545 acc: 44.21 time: 8.61
TESTING:
Accuracy of the network on the 10000 test images: 45.82 %
Average loss on the 10000 test images: 1.514
[18,   100] loss: 1.555 acc: 43.77 time: 7.18
[18,   200] loss: 1.559 acc: 44.47 time: 8.71
[18,   300] loss: 1.557 acc: 44.19 time: 7.09
TESTING:
Accuracy of the network on the 10000 test images: 45.86 %
Average loss on the 10000 test images: 1.510
[19,   100] loss: 1.556 acc: 43.67 time: 8.91
[19,   200] loss: 1.561 acc: 43.99 time: 7.15
[19,   300] loss: 1.558 acc: 44.12 time: 8.73
TESTING:
Accuracy of the network on the 10000 test images: 46.14 %
Average loss on the 10000 test images: 1.508
[20,   100] loss: 1.552 acc: 44.03 time: 7.11
[20,   200] loss: 1.544 acc: 44.70 time: 8.55
[20,   300] loss: 1.550 acc: 44.35 time: 7.16
TESTING:
Accuracy of the network on the 10000 test images: 45.74 %
Average loss on the 10000 test images: 1.507
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

# TODO: Load the pre-trained ResNet18 model
saved_state_dict = torch.load('/content/drive/MyDrive/assignment3_starter/resnet18_weights')
loaded_net2 = resnet18(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
```

```
TESTING:
Accuracy of the network on the 10000 test images: 77.51 %
Average loss on the 10000 test images: 0.580
```

```
# 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=10, init_lr=0.01, task='classification')
torch.save(loaded_net2.state_dict(), '/content/drive/MyDrive/assignment3_starter/pretrained_resnet18_all_layers_fine_tuned_weights')
```

```
[20,   200] loss: 0.362 acc: 87.79 time: 9.57
[20,   300] loss: 0.358 acc: 87.22 time: 9.75
TESTING:
Accuracy of the network on the 10000 test images: 83.84 %
Average loss on the 10000 test images: 0.486
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.

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

from torchvision.models import resnet18

# TODO: Randomly initialize a ResNet18 model
r_init_net2 = resnet18(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=10, init_lr=0.01, task='classification')
torch.save(r_init_net2.state_dict(), '/content/drive/MyDrive/assignment3_starter/random_init_resnet18_all_layers_fine_tuned_weights')
```

```
[20,   300] loss: 0.434 acc: 84.89 time: 9.21
TESTING:
Accuracy of the network on the 10000 test images: 82.21 %
Average loss on the 10000 test images: 0.527
Finished Training
```

✓ 13m 29s completed at 9:58 PM