

MIS 583 Assignment 4: Self-supervised and transfer learning on CIFAR10

Before we start, please put your name and SID in following format: : LASTNAME Firstname, ? 00000000 // e.g.) 李晨愷 M114020035

Your Answer:

Hi I'm 賴壹誠, M124020042.

Google Colab Setup

Next we need to run a few commands to set up our environment on Google Colab. If you are running this notebook on a local machine you can skip this section.

Run the following cell to mount your Google Drive. Follow the link, sign in to your Google account (the same account you used to store this notebook!) and copy the authorization code into the text box that appears below.

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

Mounted at /content/drive

Data Setup (5 points)

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

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

```
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.transforms import functional
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
#
#####
#
    if rot == 0: # 0 degrees rotation
        return img
    elif rot == 1: # 90 degrees rotation
        return transforms.functional.rotate(img, 90)
    elif rot == 2: # 180 degrees rotation
        return transforms.functional.rotate(img, 180)
    elif rot == 3: # 270 degrees rotation
        return transforms.functional.rotate(img, 270)
    else:
        raise ValueError('rotation should be 0, 90, 180, or 270
degrees')

#####
#
#      End of your code
#
#####
#

class CIFAR10Rotation(torchvision.datasets.CIFAR10):

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

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

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

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

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

transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
```

```

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

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994,
0.2010))),
    ])

batch_size = 128

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

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

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:02<00:00, 80836316.94it/s]

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

```

Show some example images and rotated images with labels:

```

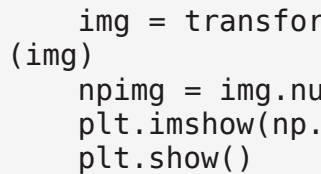
import matplotlib.pyplot as plt

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)

```

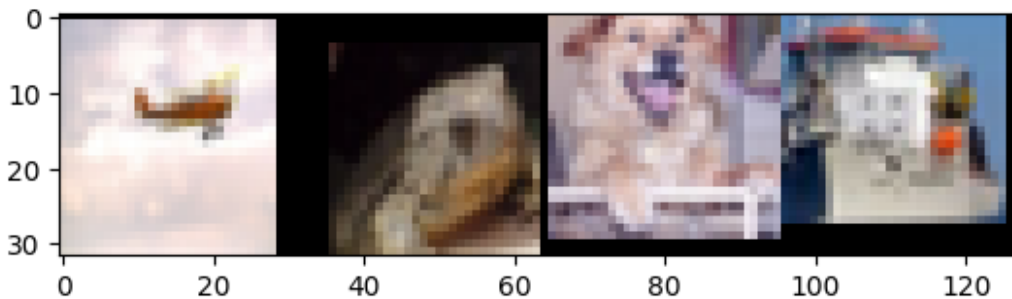
```
img = transforms.Normalize((-0.4914, -0.4822, -0.4465), (1, 1, 1))

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
```

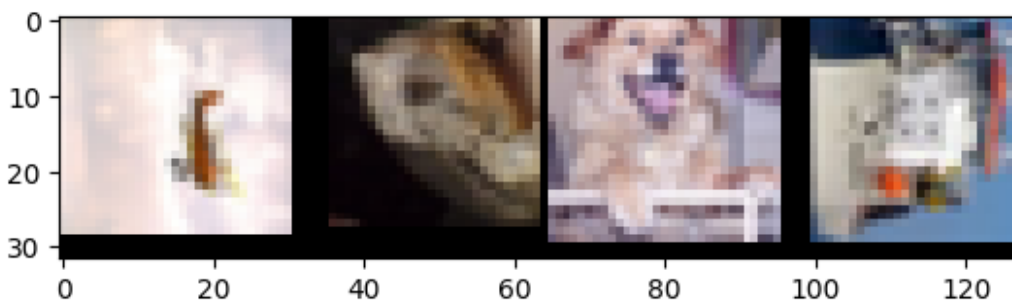
```
img_grid = imshow(torchvision.utils.make_grid(images[:4], padding=0))
print('Class labels: ', ' '.join(f'{classes[labels[j]]:5s}' for j in
range(4)))
img_grid = imshow(torchvision.utils.make_grid(rot_images[:4],
padding=0))
print('Rotation labels: ', ' '.join(f'{rot_classes[rot_labels[j]]:5s}'
for 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: plane frog dog ship



Rotation labels: 270 90 0 270

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)
#           _, predicted = torch.max(outputs.data, 1)
#           total += labels.size(0)
#           correct += (predicted == labels).sum().item()
#####
#
#           #                               End of your code
#
#####
#
#           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 (9 points)

In this section, we will train a ResNet18 model **from scratch** 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

{"type": "string"}
```

Notice: You should not use pretrained weights from ImageNet.

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

from torchvision.models import resnet18

net = resnet18(weights = None, num_classes=4) # Do not modify this line.
net = net.to(device)
print(net) # print your model and check the num_classes is correct

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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
```

```

        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
)
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
  )
)
(layer3): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
  )
)
(layer4): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    (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): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=4, bias=True)
)

import torch.nn as nn
import torch.optim as optim
#####
#####
# TODO: Define loss and optimizer functions
#
# Try any loss or optimizer function and learning rate to get better
result #
# hint: torch.nn and torch.optim
#
#####
#####
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
#####
#####
#                                     End of your code
#
#####
#####
criterion = criterion.to(device)

# 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,

```

```
#####
# 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
#
# TODO: Get predicted results
#
#####
#####
# Zero the parameter gradients
optimizer.zero_grad()

if task == 'rotation':
    # Move data to the correct device
    imgs_rotated, rotation_label =
imgs_rotated.to(device), rotation_label.to(device)
    # For rotation task, use imgs_rotated as input
    outputs = net(imgs_rotated)
    loss = criterion(outputs, rotation_label)
elif task == 'classification':
    # Move data to the correct device
    imgs, cls_label = imgs.to(device),
cls_label.to(device)
    # For classification task, use imgs as input
    outputs = net(imgs)
    loss = criterion(outputs, cls_label)
else:
    raise ValueError(f"Unsupported task: {task}")

# Backward pass and optimization
loss.backward()
optimizer.step()

# Get predicted results
_, predicted = outputs.max(1)

#####
#####
#
# End of your code
#
#####
#####
```

```

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

        # calc acc
        running_total += labels.size(0)
        running_correct += (predicted == labels.to(device)
[:predicted.size(0)]).sum().item()
        #running_correct += (predicted ==
labels.to(device)).sum().item()

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

#####
#####
        # TODO: Run the run_test() function after each epoch; Set the
model to the evaluation mode.        #

#####
#####
        run_test(net, testloader, criterion, task)

#####
#####
        #                                End of your code
#

#####
#####

        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(), 'model.pt')
#####
#        End of your code        #
#####

```

```
[1, 100] loss: 1.061 acc: 9.57 time: 9.31
[1, 200] loss: 1.052 acc: 9.53 time: 10.89
[1, 300] loss: 1.029 acc: 9.46 time: 13.57
TESTING:
Accuracy of the network on the 10000 test images: 58.22 %
Average loss on the 10000 test images: 0.996
[2, 100] loss: 1.015 acc: 9.68 time: 9.45
[2, 200] loss: 0.997 acc: 9.70 time: 10.08
[2, 300] loss: 0.994 acc: 9.91 time: 8.78
TESTING:
Accuracy of the network on the 10000 test images: 59.31 %
Average loss on the 10000 test images: 0.955
[3, 100] loss: 0.968 acc: 9.94 time: 10.51
[3, 200] loss: 0.950 acc: 9.70 time: 8.93
[3, 300] loss: 0.956 acc: 10.11 time: 10.81
TESTING:
Accuracy of the network on the 10000 test images: 61.69 %
Average loss on the 10000 test images: 0.912
[4, 100] loss: 0.927 acc: 9.82 time: 9.62
[4, 200] loss: 0.914 acc: 9.60 time: 9.97
[4, 300] loss: 0.913 acc: 9.66 time: 8.81
TESTING:
Accuracy of the network on the 10000 test images: 61.24 %
Average loss on the 10000 test images: 0.924
[5, 100] loss: 0.899 acc: 9.94 time: 10.33
[5, 200] loss: 0.896 acc: 9.77 time: 9.35
[5, 300] loss: 0.882 acc: 9.81 time: 10.15
TESTING:
Accuracy of the network on the 10000 test images: 64.57 %
Average loss on the 10000 test images: 0.850
[6, 100] loss: 0.883 acc: 9.80 time: 10.63
[6, 200] loss: 0.870 acc: 9.82 time: 9.36
[6, 300] loss: 0.866 acc: 9.63 time: 9.44
TESTING:
Accuracy of the network on the 10000 test images: 64.83 %
Average loss on the 10000 test images: 0.859
[7, 100] loss: 0.862 acc: 9.90 time: 11.06
[7, 200] loss: 0.850 acc: 9.91 time: 8.49
[7, 300] loss: 0.843 acc: 10.20 time: 10.79
TESTING:
Accuracy of the network on the 10000 test images: 65.95 %
Average loss on the 10000 test images: 0.820
[8, 100] loss: 0.825 acc: 9.70 time: 10.28
[8, 200] loss: 0.839 acc: 9.65 time: 9.92
[8, 300] loss: 0.829 acc: 9.68 time: 8.77
TESTING:
Accuracy of the network on the 10000 test images: 66.17 %
Average loss on the 10000 test images: 0.814
[9, 100] loss: 0.815 acc: 9.93 time: 8.77
[9, 200] loss: 0.825 acc: 9.89 time: 10.34
```

[9, 300] loss: 0.803 acc: 9.96 time: 9.54
TESTING:
Accuracy of the network on the 10000 test images: 67.67 %
Average loss on the 10000 test images: 0.788
[10, 100] loss: 0.791 acc: 9.84 time: 14.27
[10, 200] loss: 0.791 acc: 10.19 time: 9.72
[10, 300] loss: 0.782 acc: 9.07 time: 9.05
TESTING:
Accuracy of the network on the 10000 test images: 68.61 %
Average loss on the 10000 test images: 0.773
[11, 100] loss: 0.779 acc: 9.91 time: 9.88
[11, 200] loss: 0.763 acc: 9.66 time: 9.45
[11, 300] loss: 0.775 acc: 9.80 time: 9.96
TESTING:
Accuracy of the network on the 10000 test images: 68.60 %
Average loss on the 10000 test images: 0.768
[12, 100] loss: 0.763 acc: 10.11 time: 10.18
[12, 200] loss: 0.753 acc: 9.96 time: 9.92
[12, 300] loss: 0.753 acc: 9.41 time: 8.72
TESTING:
Accuracy of the network on the 10000 test images: 71.37 %
Average loss on the 10000 test images: 0.723
[13, 100] loss: 0.748 acc: 9.69 time: 9.70
[13, 200] loss: 0.740 acc: 10.16 time: 9.22
[13, 300] loss: 0.739 acc: 9.46 time: 10.26
TESTING:
Accuracy of the network on the 10000 test images: 70.80 %
Average loss on the 10000 test images: 0.726
[14, 100] loss: 0.724 acc: 9.70 time: 10.83
[14, 200] loss: 0.720 acc: 10.12 time: 9.55
[14, 300] loss: 0.721 acc: 9.49 time: 9.19
TESTING:
Accuracy of the network on the 10000 test images: 72.29 %
Average loss on the 10000 test images: 0.697
[15, 100] loss: 0.708 acc: 10.03 time: 9.91
[15, 200] loss: 0.711 acc: 9.62 time: 9.42
[15, 300] loss: 0.707 acc: 9.65 time: 10.33
TESTING:
Accuracy of the network on the 10000 test images: 72.01 %
Average loss on the 10000 test images: 0.697
[16, 100] loss: 0.672 acc: 9.88 time: 10.66
[16, 200] loss: 0.641 acc: 9.77 time: 10.08
[16, 300] loss: 0.641 acc: 9.79 time: 9.68
TESTING:
Accuracy of the network on the 10000 test images: 75.00 %
Average loss on the 10000 test images: 0.636
[17, 100] loss: 0.635 acc: 9.97 time: 10.81
[17, 200] loss: 0.619 acc: 9.88 time: 11.54
[17, 300] loss: 0.621 acc: 9.69 time: 11.38
TESTING:

Accuracy of the network on the 10000 test images: 74.84 %

Average loss on the 10000 test images: 0.634

[18, 100] loss: 0.611 acc: 9.96 time: 8.20

[18, 200] loss: 0.620 acc: 9.88 time: 10.91

[18, 300] loss: 0.621 acc: 9.80 time: 8.58

TESTING:

Accuracy of the network on the 10000 test images: 75.33 %

Average loss on the 10000 test images: 0.629

[19, 100] loss: 0.601 acc: 9.98 time: 11.22

[19, 200] loss: 0.605 acc: 10.06 time: 7.99

[19, 300] loss: 0.607 acc: 9.63 time: 10.85

TESTING:

Accuracy of the network on the 10000 test images: 75.42 %

Average loss on the 10000 test images: 0.616

[20, 100] loss: 0.595 acc: 9.98 time: 8.83

[20, 200] loss: 0.604 acc: 10.01 time: 10.73

[20, 300] loss: 0.600 acc: 9.97 time: 8.14

TESTING:

Accuracy of the network on the 10000 test images: 75.87 %

Average loss on the 10000 test images: 0.613

[21, 100] loss: 0.608 acc: 9.55 time: 11.31

[21, 200] loss: 0.602 acc: 10.05 time: 7.77

[21, 300] loss: 0.591 acc: 9.36 time: 11.06

TESTING:

Accuracy of the network on the 10000 test images: 76.07 %

Average loss on the 10000 test images: 0.612

[22, 100] loss: 0.589 acc: 9.96 time: 8.32

[22, 200] loss: 0.587 acc: 9.80 time: 10.81

[22, 300] loss: 0.586 acc: 10.02 time: 7.77

TESTING:

Accuracy of the network on the 10000 test images: 76.46 %

Average loss on the 10000 test images: 0.605

[23, 100] loss: 0.581 acc: 9.81 time: 11.05

[23, 200] loss: 0.586 acc: 9.62 time: 8.07

[23, 300] loss: 0.589 acc: 10.51 time: 10.90

TESTING:

Accuracy of the network on the 10000 test images: 76.05 %

Average loss on the 10000 test images: 0.605

[24, 100] loss: 0.593 acc: 9.37 time: 8.33

[24, 200] loss: 0.594 acc: 9.77 time: 10.94

[24, 300] loss: 0.578 acc: 9.62 time: 10.02

TESTING:

Accuracy of the network on the 10000 test images: 76.53 %

Average loss on the 10000 test images: 0.601

[25, 100] loss: 0.584 acc: 9.80 time: 11.05

[25, 200] loss: 0.576 acc: 9.40 time: 8.76

[25, 300] loss: 0.570 acc: 9.70 time: 10.05

TESTING:

Accuracy of the network on the 10000 test images: 76.52 %

Average loss on the 10000 test images: 0.601

```
[26, 100] loss: 0.565 acc: 9.87 time: 11.88
[26, 200] loss: 0.576 acc: 9.70 time: 10.57
[26, 300] loss: 0.583 acc: 9.39 time: 9.44
TESTING:
Accuracy of the network on the 10000 test images: 76.69 %
Average loss on the 10000 test images: 0.598
[27, 100] loss: 0.568 acc: 9.47 time: 15.03
[27, 200] loss: 0.572 acc: 9.70 time: 9.34
[27, 300] loss: 0.574 acc: 9.70 time: 9.21
TESTING:
Accuracy of the network on the 10000 test images: 77.11 %
Average loss on the 10000 test images: 0.594
[28, 100] loss: 0.562 acc: 9.73 time: 8.65
[28, 200] loss: 0.579 acc: 9.52 time: 10.44
[28, 300] loss: 0.568 acc: 9.62 time: 9.53
TESTING:
Accuracy of the network on the 10000 test images: 77.24 %
Average loss on the 10000 test images: 0.586
[29, 100] loss: 0.562 acc: 9.78 time: 12.27
[29, 200] loss: 0.575 acc: 9.77 time: 10.41
[29, 300] loss: 0.574 acc: 10.03 time: 8.36
TESTING:
Accuracy of the network on the 10000 test images: 76.77 %
Average loss on the 10000 test images: 0.598
[30, 100] loss: 0.554 acc: 9.51 time: 10.56
[30, 200] loss: 0.553 acc: 9.71 time: 9.07
[30, 300] loss: 0.561 acc: 10.06 time: 11.39
TESTING:
Accuracy of the network on the 10000 test images: 77.04 %
Average loss on the 10000 test images: 0.592
[31, 100] loss: 0.560 acc: 9.96 time: 9.03
[31, 200] loss: 0.552 acc: 10.05 time: 11.17
[31, 300] loss: 0.555 acc: 9.60 time: 8.05
TESTING:
Accuracy of the network on the 10000 test images: 77.51 %
Average loss on the 10000 test images: 0.586
[32, 100] loss: 0.557 acc: 9.84 time: 10.87
[32, 200] loss: 0.547 acc: 9.82 time: 8.06
[32, 300] loss: 0.554 acc: 9.27 time: 10.84
TESTING:
Accuracy of the network on the 10000 test images: 76.65 %
Average loss on the 10000 test images: 0.590
[33, 100] loss: 0.550 acc: 9.60 time: 8.82
[33, 200] loss: 0.557 acc: 9.91 time: 10.83
[33, 300] loss: 0.543 acc: 9.84 time: 7.75
TESTING:
Accuracy of the network on the 10000 test images: 77.37 %
Average loss on the 10000 test images: 0.585
[34, 100] loss: 0.561 acc: 9.67 time: 11.12
[34, 200] loss: 0.548 acc: 10.10 time: 7.98
```

```
[34, 300] loss: 0.537 acc: 9.71 time: 13.02
TESTING:
Accuracy of the network on the 10000 test images: 77.57 %
Average loss on the 10000 test images: 0.584
[35, 100] loss: 0.535 acc: 9.59 time: 8.05
[35, 200] loss: 0.549 acc: 9.52 time: 10.90
[35, 300] loss: 0.546 acc: 9.42 time: 9.23
TESTING:
Accuracy of the network on the 10000 test images: 77.58 %
Average loss on the 10000 test images: 0.587
[36, 100] loss: 0.540 acc: 9.83 time: 11.11
[36, 200] loss: 0.551 acc: 9.48 time: 7.93
[36, 300] loss: 0.542 acc: 9.99 time: 11.01
TESTING:
Accuracy of the network on the 10000 test images: 77.36 %
Average loss on the 10000 test images: 0.589
[37, 100] loss: 0.546 acc: 9.62 time: 8.25
[37, 200] loss: 0.552 acc: 9.80 time: 10.70
[37, 300] loss: 0.539 acc: 9.48 time: 7.72
TESTING:
Accuracy of the network on the 10000 test images: 77.23 %
Average loss on the 10000 test images: 0.584
[38, 100] loss: 0.544 acc: 9.66 time: 11.11
[38, 200] loss: 0.546 acc: 9.73 time: 7.76
[38, 300] loss: 0.541 acc: 10.03 time: 10.94
TESTING:
Accuracy of the network on the 10000 test images: 77.39 %
Average loss on the 10000 test images: 0.583
[39, 100] loss: 0.546 acc: 9.42 time: 9.13
[39, 200] loss: 0.538 acc: 9.80 time: 10.89
[39, 300] loss: 0.542 acc: 9.68 time: 8.22
TESTING:
Accuracy of the network on the 10000 test images: 77.74 %
Average loss on the 10000 test images: 0.576
[40, 100] loss: 0.547 acc: 9.66 time: 11.09
[40, 200] loss: 0.538 acc: 9.52 time: 8.27
[40, 300] loss: 0.546 acc: 9.92 time: 11.29
TESTING:
Accuracy of the network on the 10000 test images: 77.57 %
Average loss on the 10000 test images: 0.581
[41, 100] loss: 0.542 acc: 9.69 time: 9.07
[41, 200] loss: 0.550 acc: 9.78 time: 11.09
[41, 300] loss: 0.534 acc: 9.89 time: 7.84
TESTING:
Accuracy of the network on the 10000 test images: 77.67 %
Average loss on the 10000 test images: 0.582
[42, 100] loss: 0.541 acc: 10.05 time: 11.19
[42, 200] loss: 0.552 acc: 9.58 time: 8.15
[42, 300] loss: 0.543 acc: 9.63 time: 11.06
TESTING:
```



```

Accuracy of the network on the 10000 test images: 77.95 %
Average loss on the 10000 test images: 0.576
[43, 100] loss: 0.538 acc: 9.97 time: 10.70
[43, 200] loss: 0.537 acc: 10.19 time: 12.02
[43, 300] loss: 0.544 acc: 9.84 time: 8.46
TESTING:
Accuracy of the network on the 10000 test images: 77.86 %
Average loss on the 10000 test images: 0.571
[44, 100] loss: 0.536 acc: 10.03 time: 13.67
[44, 200] loss: 0.539 acc: 10.02 time: 9.43
[44, 300] loss: 0.544 acc: 9.47 time: 10.50
TESTING:
Accuracy of the network on the 10000 test images: 77.36 %
Average loss on the 10000 test images: 0.575
[45, 100] loss: 0.529 acc: 9.42 time: 9.67
[45, 200] loss: 0.540 acc: 10.09 time: 9.86
[45, 300] loss: 0.550 acc: 9.73 time: 10.05
TESTING:
Accuracy of the network on the 10000 test images: 77.93 %
Average loss on the 10000 test images: 0.571
Finished Training

```

Fine-tuning on the pre-trained model (9 points)

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

Then we will use the trained model from rotation task as the pretrained weights. Notice, you should not use the pretrained weights from ImageNet.

```

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

from torchvision.models import resnet18

#####
# TODO: Load the pre-trained ResNet18 model #
#####
ckpt = torch.load('model.pt')
net.load_state_dict(ckpt)
net.fc = nn.Linear(512, 10)
net = net.to(device)
print(net) # print your model and check the num_classes is correct
#####
#                               #
#                               #
#####

```

```

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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
)

```

```

        (1): BasicBlock(
          (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=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)
        )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),

```



```

        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 = torch.optim.SGD(params_to_update, lr=0.1, momentum=0.9)
criterion = criterion.to(device)

train(net, criterion, optimizer, num_epochs=20, decay_epochs=10,
init_lr=0.1, task='classification')

[1, 100] loss: 1.949 acc: 10.45 time: 9.14
[1, 200] loss: 1.886 acc: 10.35 time: 9.40
[1, 300] loss: 1.848 acc: 10.07 time: 7.64
TESTING:
Accuracy of the network on the 10000 test images: 32.70 %
Average loss on the 10000 test images: 1.834
[2, 100] loss: 1.773 acc: 10.09 time: 9.36
[2, 200] loss: 1.771 acc: 10.94 time: 7.74
[2, 300] loss: 1.734 acc: 10.04 time: 8.86
TESTING:
Accuracy of the network on the 10000 test images: 35.05 %
Average loss on the 10000 test images: 1.745
[3, 100] loss: 1.735 acc: 10.33 time: 11.48
[3, 200] loss: 1.692 acc: 10.02 time: 7.42
[3, 300] loss: 1.704 acc: 10.56 time: 9.51
TESTING:
Accuracy of the network on the 10000 test images: 37.49 %
Average loss on the 10000 test images: 1.714

```

```
[4, 100] loss: 1.678 acc: 10.57 time: 7.63
[4, 200] loss: 1.675 acc: 10.29 time: 9.73
[4, 300] loss: 1.674 acc: 10.19 time: 7.36
TESTING:
Accuracy of the network on the 10000 test images: 39.39 %
Average loss on the 10000 test images: 1.667
[5, 100] loss: 1.632 acc: 9.41 time: 8.90
[5, 200] loss: 1.651 acc: 10.03 time: 8.29
[5, 300] loss: 1.651 acc: 10.27 time: 8.49
TESTING:
Accuracy of the network on the 10000 test images: 41.74 %
Average loss on the 10000 test images: 1.621
[6, 100] loss: 1.607 acc: 10.26 time: 10.32
[6, 200] loss: 1.611 acc: 10.04 time: 7.04
[6, 300] loss: 1.603 acc: 10.34 time: 9.88
TESTING:
Accuracy of the network on the 10000 test images: 40.66 %
Average loss on the 10000 test images: 1.613
[7, 100] loss: 1.569 acc: 10.07 time: 8.26
[7, 200] loss: 1.552 acc: 10.23 time: 8.58
[7, 300] loss: 1.551 acc: 9.74 time: 8.37
TESTING:
Accuracy of the network on the 10000 test images: 43.35 %
Average loss on the 10000 test images: 1.550
[8, 100] loss: 1.538 acc: 10.40 time: 7.31
[8, 200] loss: 1.541 acc: 10.18 time: 10.00
[8, 300] loss: 1.523 acc: 10.33 time: 7.42
TESTING:
Accuracy of the network on the 10000 test images: 44.52 %
Average loss on the 10000 test images: 1.525
[9, 100] loss: 1.509 acc: 9.98 time: 8.92
[9, 200] loss: 1.514 acc: 9.85 time: 8.09
[9, 300] loss: 1.493 acc: 10.62 time: 8.82
TESTING:
Accuracy of the network on the 10000 test images: 44.18 %
Average loss on the 10000 test images: 1.523
[10, 100] loss: 1.494 acc: 10.07 time: 9.92
[10, 200] loss: 1.491 acc: 10.41 time: 7.19
[10, 300] loss: 1.478 acc: 10.00 time: 9.86
TESTING:
Accuracy of the network on the 10000 test images: 43.62 %
Average loss on the 10000 test images: 1.525
[11, 100] loss: 1.430 acc: 10.23 time: 10.54
[11, 200] loss: 1.431 acc: 9.98 time: 7.33
[11, 300] loss: 1.412 acc: 10.27 time: 9.56
TESTING:
Accuracy of the network on the 10000 test images: 47.16 %
Average loss on the 10000 test images: 1.439
[12, 100] loss: 1.419 acc: 9.53 time: 7.73
```

```
[12, 200] loss: 1.408 acc: 10.30 time: 9.61
[12, 300] loss: 1.405 acc: 10.04 time: 7.27
TESTING:
Accuracy of the network on the 10000 test images: 47.63 %
Average loss on the 10000 test images: 1.431
[13, 100] loss: 1.388 acc: 10.51 time: 7.55
[13, 200] loss: 1.406 acc: 9.96 time: 9.47
[13, 300] loss: 1.410 acc: 10.26 time: 7.18
TESTING:
Accuracy of the network on the 10000 test images: 47.96 %
Average loss on the 10000 test images: 1.424
[14, 100] loss: 1.405 acc: 9.88 time: 9.78
[14, 200] loss: 1.394 acc: 9.47 time: 7.47
[14, 300] loss: 1.395 acc: 10.39 time: 9.70
TESTING:
Accuracy of the network on the 10000 test images: 48.57 %
Average loss on the 10000 test images: 1.420
[15, 100] loss: 1.378 acc: 10.12 time: 9.65
[15, 200] loss: 1.390 acc: 10.28 time: 7.67
[15, 300] loss: 1.388 acc: 10.09 time: 9.63
TESTING:
Accuracy of the network on the 10000 test images: 48.62 %
Average loss on the 10000 test images: 1.414
[16, 100] loss: 1.381 acc: 10.30 time: 7.48
[16, 200] loss: 1.377 acc: 9.98 time: 9.86
[16, 300] loss: 1.392 acc: 9.89 time: 7.26
TESTING:
Accuracy of the network on the 10000 test images: 48.89 %
Average loss on the 10000 test images: 1.405
[17, 100] loss: 1.407 acc: 10.15 time: 8.02
[17, 200] loss: 1.367 acc: 10.05 time: 9.33
[17, 300] loss: 1.391 acc: 9.91 time: 7.76
TESTING:
Accuracy of the network on the 10000 test images: 48.93 %
Average loss on the 10000 test images: 1.404
[18, 100] loss: 1.383 acc: 10.25 time: 13.02
[18, 200] loss: 1.389 acc: 9.73 time: 7.44
[18, 300] loss: 1.383 acc: 10.26 time: 9.86
TESTING:
Accuracy of the network on the 10000 test images: 48.90 %
Average loss on the 10000 test images: 1.395
[19, 100] loss: 1.377 acc: 9.67 time: 7.88
[19, 200] loss: 1.377 acc: 10.04 time: 9.68
[19, 300] loss: 1.378 acc: 9.95 time: 7.69
TESTING:
Accuracy of the network on the 10000 test images: 49.16 %
Average loss on the 10000 test images: 1.394
[20, 100] loss: 1.376 acc: 10.42 time: 8.91
[20, 200] loss: 1.380 acc: 9.74 time: 11.80
```

```
[20, 300] loss: 1.377 acc: 9.71 time: 7.37
TESTING:
Accuracy of the network on the 10000 test images: 49.44 %
Average loss on the 10000 test images: 1.392
Finished Training
```

Fine-tuning on the randomly initialized model (9 points)

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 #
#####
net = resnet18(weights=None, num_classes=10)
net = net.to(device)
print(net) # print your model and check the num_classes is correct
#####
#                               End of your code                               #
#####

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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
```



```

track_running_stats=True)
    (relu): ReLU(inplace=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)
    )
    )
    (layer2): Sequential(
    (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
        (downsample): Sequential(
            (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
    )
    (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
    )
    (layer3): Sequential(
    (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
    )

```

```

        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
        (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): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
      )
    )
  )
)

```

```

    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=10, bias=True)
)

#####
#####
# 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 name, param in net.named_parameters():
    if 'layer4' in name or 'fc' in name:
        param.requires_grad = True
    else:
        param.requires_grad = False
#####
#####
#
#                                     End of your code
#
#####
#####

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

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

```

```
fc.weight
fc.bias
```

```
# TODO: Define criterion and optimizer
```

```
# Note that your optimizer only needs to update the parameters that  
are trainable.
```

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(params_to_update, lr=0.1, momentum=0.9)
```

```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=10,  
init_lr=0.1, task='classification')
```

```
[1, 100] loss: 1.586 acc: 10.27 time: 6.76
```

```
[1, 200] loss: 1.650 acc: 9.96 time: 8.43
```

```
[1, 300] loss: 1.635 acc: 10.45 time: 9.23
```

```
TESTING:
```

```
Accuracy of the network on the 10000 test images: 41.49 %
```

```
Average loss on the 10000 test images: 1.651
```

```
[2, 100] loss: 1.634 acc: 9.67 time: 8.65
```

```
[2, 200] loss: 1.629 acc: 10.16 time: 6.57
```

```
[2, 300] loss: 1.635 acc: 10.16 time: 10.40
```

```
TESTING:
```

```
Accuracy of the network on the 10000 test images: 42.60 %
```

```
Average loss on the 10000 test images: 1.632
```

```
[3, 100] loss: 1.623 acc: 9.93 time: 7.44
```

```
[3, 200] loss: 1.614 acc: 9.79 time: 7.72
```

```
[3, 300] loss: 1.609 acc: 10.38 time: 7.27
```

```
TESTING:
```

```
Accuracy of the network on the 10000 test images: 42.35 %
```

```
Average loss on the 10000 test images: 1.639
```

```
[4, 100] loss: 1.595 acc: 9.74 time: 6.77
```

```
[4, 200] loss: 1.622 acc: 9.85 time: 8.08
```

```
[4, 300] loss: 1.607 acc: 10.01 time: 6.67
```

```
TESTING:
```

```
Accuracy of the network on the 10000 test images: 42.77 %
```

```
Average loss on the 10000 test images: 1.618
```

```
[5, 100] loss: 1.598 acc: 10.24 time: 8.47
```

```
[5, 200] loss: 1.588 acc: 10.10 time: 6.46
```

```
[5, 300] loss: 1.624 acc: 10.33 time: 8.38
```

```
TESTING:
```

```
Accuracy of the network on the 10000 test images: 43.27 %
```

```
Average loss on the 10000 test images: 1.606
```

```
[6, 100] loss: 1.593 acc: 9.98 time: 7.75
```

```
[6, 200] loss: 1.591 acc: 10.00 time: 7.46
```

```
[6, 300] loss: 1.594 acc: 9.62 time: 7.44
```

```
TESTING:
```

```
Accuracy of the network on the 10000 test images: 43.83 %
```

```
Average loss on the 10000 test images: 1.595
```

```
[7, 100] loss: 1.592 acc: 10.00 time: 6.69
```

```
[7, 200] loss: 1.576 acc: 10.02 time: 8.49
```

[7, 300] loss: 1.587 acc: 10.42 time: 6.65
TESTING:
Accuracy of the network on the 10000 test images: 43.51 %
Average loss on the 10000 test images: 1.600
[8, 100] loss: 1.577 acc: 10.48 time: 8.35
[8, 200] loss: 1.575 acc: 10.24 time: 6.59
[8, 300] loss: 1.582 acc: 10.10 time: 8.32
TESTING:
Accuracy of the network on the 10000 test images: 42.86 %
Average loss on the 10000 test images: 1.609
[9, 100] loss: 1.567 acc: 10.52 time: 8.09
[9, 200] loss: 1.571 acc: 9.96 time: 7.69
[9, 300] loss: 1.578 acc: 9.92 time: 7.56
TESTING:
Accuracy of the network on the 10000 test images: 43.84 %
Average loss on the 10000 test images: 1.588
[10, 100] loss: 1.569 acc: 10.36 time: 6.51
[10, 200] loss: 1.558 acc: 10.29 time: 8.38
[10, 300] loss: 1.573 acc: 10.26 time: 6.57
TESTING:
Accuracy of the network on the 10000 test images: 43.44 %
Average loss on the 10000 test images: 1.597
[11, 100] loss: 1.553 acc: 9.68 time: 8.11
[11, 200] loss: 1.525 acc: 9.91 time: 6.91
[11, 300] loss: 1.520 acc: 10.13 time: 7.58
TESTING:
Accuracy of the network on the 10000 test images: 44.79 %
Average loss on the 10000 test images: 1.568
[12, 100] loss: 1.525 acc: 10.12 time: 8.54
[12, 200] loss: 1.512 acc: 10.27 time: 7.74
[12, 300] loss: 1.515 acc: 10.03 time: 8.72
TESTING:
Accuracy of the network on the 10000 test images: 44.92 %
Average loss on the 10000 test images: 1.556
[13, 100] loss: 1.500 acc: 10.34 time: 6.68
[13, 200] loss: 1.505 acc: 9.66 time: 8.09
[13, 300] loss: 1.519 acc: 10.39 time: 8.13
TESTING:
Accuracy of the network on the 10000 test images: 45.32 %
Average loss on the 10000 test images: 1.553
[14, 100] loss: 1.503 acc: 10.20 time: 8.14
[14, 200] loss: 1.491 acc: 10.06 time: 6.74
[14, 300] loss: 1.507 acc: 10.06 time: 8.19
TESTING:
Accuracy of the network on the 10000 test images: 45.27 %
Average loss on the 10000 test images: 1.555
[15, 100] loss: 1.492 acc: 10.12 time: 8.36
[15, 200] loss: 1.509 acc: 10.23 time: 6.46
[15, 300] loss: 1.497 acc: 10.49 time: 8.32

```

TESTING:
Accuracy of the network on the 10000 test images: 45.26 %
Average loss on the 10000 test images: 1.549
[16, 100] loss: 1.498 acc: 9.76 time: 6.95
[16, 200] loss: 1.506 acc: 10.10 time: 8.17
[16, 300] loss: 1.485 acc: 10.27 time: 6.52
TESTING:
Accuracy of the network on the 10000 test images: 45.22 %
Average loss on the 10000 test images: 1.547
[17, 100] loss: 1.486 acc: 10.08 time: 6.99
[17, 200] loss: 1.493 acc: 9.98 time: 7.79
[17, 300] loss: 1.499 acc: 10.12 time: 6.93
TESTING:
Accuracy of the network on the 10000 test images: 45.45 %
Average loss on the 10000 test images: 1.547
[18, 100] loss: 1.491 acc: 10.15 time: 8.27
[18, 200] loss: 1.489 acc: 9.92 time: 6.56
[18, 300] loss: 1.484 acc: 9.72 time: 8.12
TESTING:
Accuracy of the network on the 10000 test images: 45.70 %
Average loss on the 10000 test images: 1.545
[19, 100] loss: 1.479 acc: 10.12 time: 7.51
[19, 200] loss: 1.477 acc: 10.17 time: 7.59
[19, 300] loss: 1.494 acc: 9.76 time: 7.20
TESTING:
Accuracy of the network on the 10000 test images: 45.48 %
Average loss on the 10000 test images: 1.540
[20, 100] loss: 1.483 acc: 9.68 time: 6.73
[20, 200] loss: 1.481 acc: 10.14 time: 8.19
[20, 300] loss: 1.494 acc: 9.82 time: 6.34
TESTING:
Accuracy of the network on the 10000 test images: 45.82 %
Average loss on the 10000 test images: 1.539
Finished Training

```

Supervised training on the pre-trained model (9 points)

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

Then we will use the trained model from rotation task as the pretrained weights. Notice, you should not use the pretrained weights from ImageNet.

```

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

from torchvision.models import resnet18

#####

```

```

#      TODO: Load the pre-trained ResNet18 model      #
#####
ckpt = torch.load('model.pt')
net.load_state_dict(ckpt)
net.fc = nn.Linear(512, 10)
net = net.to(device)
print(net) # print your model and check the num_classes is correct
#####
#      End of your code      #
#####

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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
  )
(layer3): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
        (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): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=10, bias=True)
  )

```

TODO: *Define criterion and optimizer*

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
```

```
train(net, criterion, optimizer, num_epochs=20, decay_epochs=10,
init_lr=0.1, task='classification')
```

```
[1, 100] loss: 1.141 acc: 9.93 time: 9.30
```

```
[1, 200] loss: 1.141 acc: 10.26 time: 8.80
```

```
[1, 300] loss: 1.124 acc: 10.27 time: 8.80
```

TESTING:

Accuracy of the network on the 10000 test images: 58.98 %

Average loss on the 10000 test images: 1.139
[2, 100] loss: 1.096 acc: 9.85 time: 7.67
[2, 200] loss: 1.110 acc: 10.02 time: 10.13
[2, 300] loss: 1.076 acc: 9.84 time: 7.64
TESTING:
Accuracy of the network on the 10000 test images: 59.25 %
Average loss on the 10000 test images: 1.143
[3, 100] loss: 1.077 acc: 9.98 time: 10.37
[3, 200] loss: 1.076 acc: 9.48 time: 7.52
[3, 300] loss: 1.080 acc: 10.30 time: 10.04
TESTING:
Accuracy of the network on the 10000 test images: 61.30 %
Average loss on the 10000 test images: 1.092
[4, 100] loss: 1.055 acc: 9.77 time: 9.33
[4, 200] loss: 1.022 acc: 10.15 time: 9.17
[4, 300] loss: 1.032 acc: 9.99 time: 8.67
TESTING:
Accuracy of the network on the 10000 test images: 62.23 %
Average loss on the 10000 test images: 1.059
[5, 100] loss: 1.032 acc: 10.37 time: 7.86
[5, 200] loss: 1.024 acc: 10.20 time: 10.22
[5, 300] loss: 1.018 acc: 10.03 time: 8.26
TESTING:
Accuracy of the network on the 10000 test images: 62.74 %
Average loss on the 10000 test images: 1.050
[6, 100] loss: 1.000 acc: 9.84 time: 10.81
[6, 200] loss: 1.000 acc: 9.85 time: 10.85
[6, 300] loss: 0.985 acc: 9.91 time: 9.41
TESTING:
Accuracy of the network on the 10000 test images: 64.15 %
Average loss on the 10000 test images: 1.009
[7, 100] loss: 0.972 acc: 10.46 time: 8.46
[7, 200] loss: 0.972 acc: 10.24 time: 8.81
[7, 300] loss: 0.976 acc: 10.24 time: 9.54
TESTING:
Accuracy of the network on the 10000 test images: 64.63 %
Average loss on the 10000 test images: 0.993
[8, 100] loss: 0.948 acc: 10.29 time: 9.65
[8, 200] loss: 0.967 acc: 9.99 time: 8.70
[8, 300] loss: 0.963 acc: 9.97 time: 8.58
TESTING:
Accuracy of the network on the 10000 test images: 64.75 %
Average loss on the 10000 test images: 0.995
[9, 100] loss: 0.925 acc: 10.66 time: 8.14
[9, 200] loss: 0.949 acc: 9.90 time: 9.05
[9, 300] loss: 0.948 acc: 9.83 time: 11.01
TESTING:
Accuracy of the network on the 10000 test images: 65.75 %
Average loss on the 10000 test images: 0.961

```
[10, 100] loss: 0.928 acc: 10.55 time: 9.87
[10, 200] loss: 0.933 acc: 9.79 time: 7.67
[10, 300] loss: 0.917 acc: 9.64 time: 9.06
TESTING:
Accuracy of the network on the 10000 test images: 66.56 %
Average loss on the 10000 test images: 0.946
[11, 100] loss: 0.877 acc: 10.20 time: 7.68
[11, 200] loss: 0.861 acc: 9.82 time: 9.53
[11, 300] loss: 0.870 acc: 9.92 time: 8.15
TESTING:
Accuracy of the network on the 10000 test images: 67.46 %
Average loss on the 10000 test images: 0.921
[12, 100] loss: 0.864 acc: 9.96 time: 9.74
[12, 200] loss: 0.863 acc: 10.36 time: 7.89
[12, 300] loss: 0.858 acc: 10.02 time: 9.49
TESTING:
Accuracy of the network on the 10000 test images: 67.75 %
Average loss on the 10000 test images: 0.917
[13, 100] loss: 0.855 acc: 10.31 time: 7.56
[13, 200] loss: 0.858 acc: 10.17 time: 9.46
[13, 300] loss: 0.855 acc: 10.28 time: 7.76
TESTING:
Accuracy of the network on the 10000 test images: 67.68 %
Average loss on the 10000 test images: 0.917
[14, 100] loss: 0.853 acc: 10.20 time: 9.77
[14, 200] loss: 0.868 acc: 9.66 time: 7.43
[14, 300] loss: 0.849 acc: 10.10 time: 9.53
TESTING:
Accuracy of the network on the 10000 test images: 67.69 %
Average loss on the 10000 test images: 0.918
[15, 100] loss: 0.846 acc: 9.77 time: 7.42
[15, 200] loss: 0.871 acc: 10.17 time: 9.60
[15, 300] loss: 0.858 acc: 9.53 time: 7.50
TESTING:
Accuracy of the network on the 10000 test images: 67.74 %
Average loss on the 10000 test images: 0.916
[16, 100] loss: 0.866 acc: 10.02 time: 9.69
[16, 200] loss: 0.857 acc: 9.88 time: 7.47
[16, 300] loss: 0.841 acc: 10.24 time: 9.72
TESTING:
Accuracy of the network on the 10000 test images: 68.07 %
Average loss on the 10000 test images: 0.912
[17, 100] loss: 0.835 acc: 9.73 time: 9.83
[17, 200] loss: 0.837 acc: 10.26 time: 9.52
[17, 300] loss: 0.868 acc: 10.08 time: 7.40
TESTING:
Accuracy of the network on the 10000 test images: 67.95 %
Average loss on the 10000 test images: 0.914
[18, 100] loss: 0.848 acc: 9.73 time: 9.86
```

```

[18, 200] loss: 0.840 acc: 9.76 time: 7.77
[18, 300] loss: 0.859 acc: 10.43 time: 9.85
TESTING:
Accuracy of the network on the 10000 test images: 67.81 %
Average loss on the 10000 test images: 0.911
[19, 100] loss: 0.846 acc: 9.70 time: 7.83
[19, 200] loss: 0.837 acc: 9.47 time: 9.88
[19, 300] loss: 0.849 acc: 10.04 time: 9.76
TESTING:
Accuracy of the network on the 10000 test images: 67.84 %
Average loss on the 10000 test images: 0.914
[20, 100] loss: 0.846 acc: 10.00 time: 9.87
[20, 200] loss: 0.857 acc: 10.48 time: 7.50
[20, 300] loss: 0.860 acc: 10.04 time: 9.70
TESTING:
Accuracy of the network on the 10000 test images: 67.94 %
Average loss on the 10000 test images: 0.909
Finished Training

```

Supervised training on the randomly initialized model (9 points)

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 #
#####
net = resnet18(weights=None, num_classes=10)
net = net.to(device)
print(net) # print your model and check the num_classes is correct
#####
#                               End of your code                               #
#####

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): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
  )
  (1): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
  )
)
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=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)
    )
    )
    (layer3): Sequential(
    (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
        (downsample): Sequential(
            (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
    )
    (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
    )
    (layer4): Sequential(
    (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
        (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): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=10, bias=True)
)

```

TODO: Define criterion and optimizer

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
```

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

```
[1, 100] loss: 1.999 acc: 10.26 time: 17.28
```

```
[1, 200] loss: 1.728 acc: 10.05 time: 10.00
```

```
[1, 300] loss: 1.579 acc: 10.00 time: 8.26
```

TESTING:

Accuracy of the network on the 10000 test images: 49.51 %

Average loss on the 10000 test images: 1.389

```
[2, 100] loss: 1.436 acc: 9.73 time: 8.72
```

```
[2, 200] loss: 1.351 acc: 9.71 time: 10.36
```

```
[2, 300] loss: 1.322 acc: 10.11 time: 10.74
```

TESTING:

Accuracy of the network on the 10000 test images: 57.21 %

Average loss on the 10000 test images: 1.181

```
[3, 100] loss: 1.230 acc: 10.23 time: 11.44
```

```
[3, 200] loss: 1.175 acc: 9.95 time: 7.98
```

```
[3, 300] loss: 1.178 acc: 10.21 time: 12.58
```

TESTING:

Accuracy of the network on the 10000 test images: 61.77 %

Average loss on the 10000 test images: 1.078

```
[4, 100] loss: 1.091 acc: 10.52 time: 8.06
```

```
[4, 200] loss: 1.085 acc: 9.88 time: 10.73
```

```
[4, 300] loss: 1.047 acc: 10.16 time: 8.33
```

TESTING:

Accuracy of the network on the 10000 test images: 64.13 %

Average loss on the 10000 test images: 1.017
[5, 100] loss: 1.019 acc: 10.27 time: 10.97
[5, 200] loss: 0.993 acc: 9.38 time: 7.73
[5, 300] loss: 0.954 acc: 9.67 time: 10.87
TESTING:
Accuracy of the network on the 10000 test images: 68.41 %
Average loss on the 10000 test images: 0.906
[6, 100] loss: 0.943 acc: 10.19 time: 9.50
[6, 300] loss: 0.916 acc: 9.86 time: 8.07
TESTING:
Accuracy of the network on the 10000 test images: 69.97 %
Average loss on the 10000 test images: 0.868
[7, 100] loss: 0.886 acc: 10.55 time: 9.33
[7, 200] loss: 0.871 acc: 9.84 time: 9.39
[7, 300] loss: 0.869 acc: 10.59 time: 9.36
TESTING:
Accuracy of the network on the 10000 test images: 70.37 %
Average loss on the 10000 test images: 0.841
[8, 100] loss: 0.820 acc: 10.12 time: 10.89
[8, 200] loss: 0.826 acc: 9.99 time: 7.69
[8, 300] loss: 0.812 acc: 9.80 time: 11.07
TESTING:
Accuracy of the network on the 10000 test images: 71.84 %
Average loss on the 10000 test images: 0.805
[9, 100] loss: 0.785 acc: 9.72 time: 8.13
[9, 200] loss: 0.778 acc: 10.43 time: 10.70
[9, 300] loss: 0.758 acc: 10.16 time: 7.62
TESTING:
Accuracy of the network on the 10000 test images: 72.88 %
Average loss on the 10000 test images: 0.803
[10, 100] loss: 0.745 acc: 9.59 time: 10.17
[10, 200] loss: 0.752 acc: 10.18 time: 10.74
[10, 300] loss: 0.739 acc: 9.91 time: 10.73
TESTING:
Accuracy of the network on the 10000 test images: 73.79 %
Average loss on the 10000 test images: 0.770
[11, 100] loss: 0.663 acc: 10.52 time: 9.08
[11, 200] loss: 0.644 acc: 10.48 time: 10.70
[11, 300] loss: 0.649 acc: 10.17 time: 8.17
TESTING:
Accuracy of the network on the 10000 test images: 75.89 %
Average loss on the 10000 test images: 0.701
[12, 100] loss: 0.615 acc: 10.48 time: 9.40
[12, 200] loss: 0.620 acc: 10.32 time: 9.24
[12, 300] loss: 0.608 acc: 10.01 time: 9.50
TESTING:
Accuracy of the network on the 10000 test images: 76.17 %
Average loss on the 10000 test images: 0.698
[13, 100] loss: 0.619 acc: 10.20 time: 10.91


```
[13, 200] loss: 0.608 acc: 9.62 time: 8.62
[13, 300] loss: 0.609 acc: 9.91 time: 10.02
TESTING:
Accuracy of the network on the 10000 test images: 76.57 %
Average loss on the 10000 test images: 0.697
[14, 100] loss: 0.610 acc: 9.99 time: 8.06
[14, 200] loss: 0.598 acc: 10.59 time: 10.69
[14, 300] loss: 0.587 acc: 9.87 time: 10.50
TESTING:
Accuracy of the network on the 10000 test images: 76.26 %
Average loss on the 10000 test images: 0.698
[15, 100] loss: 0.580 acc: 10.16 time: 10.83
[15, 200] loss: 0.594 acc: 9.97 time: 7.84
[15, 300] loss: 0.586 acc: 10.38 time: 10.76
TESTING:
Accuracy of the network on the 10000 test images: 76.81 %
Average loss on the 10000 test images: 0.687
[16, 100] loss: 0.590 acc: 9.71 time: 9.62
[16, 200] loss: 0.587 acc: 10.11 time: 9.63
[16, 300] loss: 0.579 acc: 10.11 time: 8.87
TESTING:
Accuracy of the network on the 10000 test images: 76.70 %
Average loss on the 10000 test images: 0.688
[17, 100] loss: 0.575 acc: 10.13 time: 8.50
[17, 200] loss: 0.566 acc: 9.65 time: 10.21
[17, 300] loss: 0.570 acc: 9.90 time: 8.60
TESTING:
Accuracy of the network on the 10000 test images: 76.72 %
Average loss on the 10000 test images: 0.682
[18, 100] loss: 0.560 acc: 10.05 time: 10.93
[18, 200] loss: 0.562 acc: 10.39 time: 7.68
[18, 300] loss: 0.565 acc: 9.79 time: 10.56
TESTING:
Accuracy of the network on the 10000 test images: 77.23 %
Average loss on the 10000 test images: 0.678
[19, 100] loss: 0.564 acc: 9.80 time: 8.38
[19, 200] loss: 0.560 acc: 9.79 time: 10.30
[19, 300] loss: 0.562 acc: 10.25 time: 8.02
TESTING:
Accuracy of the network on the 10000 test images: 77.14 %
Average loss on the 10000 test images: 0.678
[20, 100] loss: 0.550 acc: 10.40 time: 9.20
[20, 200] loss: 0.550 acc: 10.00 time: 9.80
[20, 300] loss: 0.561 acc: 10.38 time: 9.36
TESTING:
Accuracy of the network on the 10000 test images: 77.10 %
Average loss on the 10000 test images: 0.677
Finished Training
```

Write report (37 points)

本次作業主要有 3 個 tasks 需要大家完成，在 A4.pdf 中有希望大家達成的 baseline (不能低於 baseline 最多 2%，沒有達到不會給全部分數)，report 的撰寫請大家根據以下要求完成，就請大家將嘗試的結果寫在 report 裡，祝大家順利！

1. (13 points) Train a ResNet18 on the Rotation task and report the test performance. Discuss why such a task helps in learning features that are generalizable to other visual tasks.
2. (12 points) Initializing from the Rotation model or from random weights, fine-tune only the weights of the final block of convolutional layers and linear layer on the supervised CIFAR10 classification task. Report the test results and compare the performance of these two models. Provide your observations and insights. You can also discuss how the performance of pre-trained models affects downstream tasks, the performance of fine-tuning different numbers of layers, and so on.
3. (12 points) Initializing from the Rotation model or from random weights, train the full network on the supervised CIFAR10 classification task. Report the test results and compare the performance of these two models. Provide your observations and insights.

Extra Credit (13 points)

上面基本的 code 跟 report 最高可以拿到 87 分，這個加分部分並沒有要求同學們一定要做，若同學們想要獲得更高的分數可以根據以下的加分要求來獲得加分。

- In Figure 5(b) from the Gidaris et al. paper, the authors show a plot of CIFAR10 classification performance vs. number of training examples per category for a supervised CIFAR10 model vs. a RotNet model with the final layers fine-tuned on CIFAR10. The plot shows that pre-training on the Rotation task can be advantageous when only a small amount of labeled data is available. Using your RotNet fine-tuning code and supervised CIFAR10 training code from the main assignment, try to create a similar plot by performing supervised fine-tuning/training on only a subset of CIFAR10.
- Use a more advanced model than ResNet18 to try to get higher accuracy on the rotation prediction task, as well as for transfer to supervised CIFAR10 classification.
- If you have a good amount of compute at your disposal, try to train a rotation prediction model on the larger ImageNet dataset (still smaller than ImageNet, though).