(Optional) Colab Setup

If you aren't using Colab, you can delete the following code cell. This is just to help students with mounting to Google Drive to access the other .py files and downloading the data, which is a little trickier on Colab than on your local machine using Jupyter.

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

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

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

```
/content/drive/MyDrive/assignment3
```

Data Setup

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

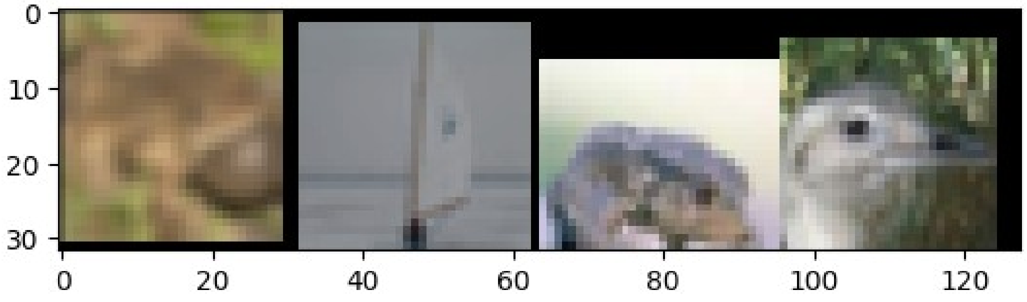
Each rotation label should be an integer in the set {0, 1, 2, 3} which correspond to rotations of 0, 90, 180, or 270 degrees respectively.

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


def rotate_img(img, rot):
    if rot == 0 : # 0 degrees rotation
        return img
    # TODO: Implement rotate_img() - return the rotated img
    elif rot == 1 :
        return transforms.functional.rotate(img,90)
    elif rot == 2 :
        return transforms.functional.rotate(img,180)
    elif rot == 3 :
        return transforms.functional.rotate(img,270)
    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_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomApply([transforms.ColorJitter(0.2, 0.3, 0.1,
0.2)], p=0.3),
    transforms.RandomApply([transforms.GaussianBlur(3)], p=0.2),
    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=0)

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=0)
```

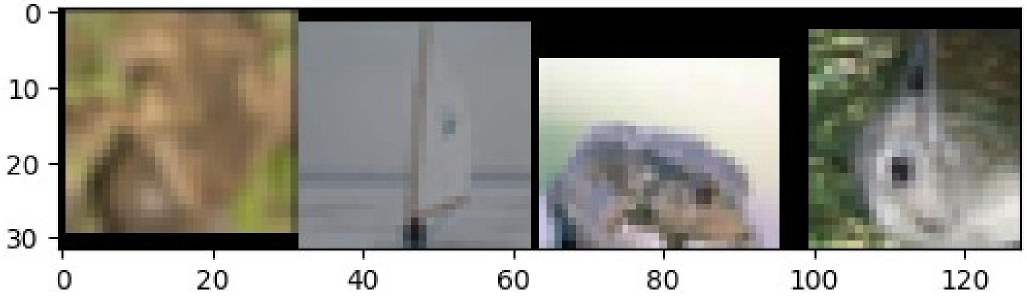
Show some example images and rotated images with labels:

```
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)))
```

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-2.9802322e-
08..0.9882353].
```



```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-2.9802322e-
08..0.9882353].

Class labels:  frog  ship  frog  bird
```



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

Evaluation code

```
device = 'mps'

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 = torch.max(outputs,dim=1).indices
            total += labels.shape[0]
            correct += (predictions==labels).sum().item()

            # 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
```

---

1. 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.

```
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

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

# 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)

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

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

            # TODO: Get predicted results
            predicted = torch.max(outputs,dim=1).indices

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

            # calc acc
            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()

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

        # TODO: Run the run_test() function after each epoch; Set the
model to the evaluation mode.
        net.eval()
        run_test(net,testloader,criterion,task)

    print('Finished Training')

checkpoint =
torch.load("./models/resnet18_rotation.pth",map_location=torch.device(
'mps'))
net.load_state_dict(checkpoint['parameters'])
optimizer.load_state_dict(checkpoint['optimizer'])

train(net, criterion, optimizer, num_epochs=50, decay_epochs=15,
init_lr=0.001, task='rotation')

print('Saving Model ...')
# TODO: Save the model
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()}, "./models/resnet18_rotation.pth")
print('Saved Model !')
```

```
[1,   100] loss: 1.281 acc: 42.84 time: 3.62
[1,   200] loss: 1.119 acc: 50.42 time: 3.52
[1,   300] loss: 1.107 acc: 51.63 time: 3.52
TESTING:
Accuracy of the network on the 10000 test images: 56.04 %
Average loss on the 10000 test images: 1.037
[2,   100] loss: 1.048 acc: 55.14 time: 3.54
[2,   200] loss: 1.034 acc: 56.01 time: 3.53
[2,   300] loss: 1.005 acc: 56.76 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 59.25 %
Average loss on the 10000 test images: 0.972
```

```
[3,   100] loss: 0.979 acc: 59.07 time: 3.54
[3,   200] loss: 0.960 acc: 59.30 time: 3.56
[3,   300] loss: 0.954 acc: 59.70 time: 3.57
TESTING:
Accuracy of the network on the 10000 test images: 62.54 %
Average loss on the 10000 test images: 0.895
[4,   100] loss: 0.946 acc: 60.52 time: 3.59
[4,   200] loss: 0.911 acc: 62.16 time: 3.60
[4,   300] loss: 0.926 acc: 61.02 time: 3.56
TESTING:
Accuracy of the network on the 10000 test images: 62.97 %
Average loss on the 10000 test images: 0.897
[5,   100] loss: 0.905 acc: 62.94 time: 3.54
[5,   200] loss: 0.884 acc: 63.34 time: 3.55
[5,   300] loss: 0.894 acc: 63.06 time: 3.56
TESTING:
Accuracy of the network on the 10000 test images: 63.12 %
Average loss on the 10000 test images: 0.896
[6,   100] loss: 0.858 acc: 64.66 time: 3.61
[6,   200] loss: 0.869 acc: 64.12 time: 3.54
[6,   300] loss: 0.848 acc: 64.71 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 66.04 %
Average loss on the 10000 test images: 0.823
[7,   100] loss: 0.844 acc: 65.26 time: 3.55
[7,   200] loss: 0.834 acc: 65.89 time: 3.54
[7,   300] loss: 0.835 acc: 65.83 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 65.45 %
Average loss on the 10000 test images: 0.830
[8,   100] loss: 0.825 acc: 66.20 time: 3.54
[8,   200] loss: 0.821 acc: 66.84 time: 3.54
[8,   300] loss: 0.807 acc: 67.23 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 68.47 %
Average loss on the 10000 test images: 0.773
[9,   100] loss: 0.784 acc: 67.95 time: 3.56
[9,   200] loss: 0.792 acc: 68.07 time: 3.53
[9,   300] loss: 0.787 acc: 67.76 time: 3.63
TESTING:
Accuracy of the network on the 10000 test images: 69.51 %
Average loss on the 10000 test images: 0.752
[10,   100] loss: 0.794 acc: 68.12 time: 3.55
[10,   200] loss: 0.769 acc: 68.99 time: 3.54
[10,   300] loss: 0.771 acc: 68.30 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 69.77 %
Average loss on the 10000 test images: 0.759
[11,   100] loss: 0.772 acc: 68.81 time: 3.54
```

```
[11,   200] loss: 0.762 acc: 69.68 time: 3.54
[11,   300] loss: 0.741 acc: 70.27 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 70.34 %
Average loss on the 10000 test images: 0.739
[12,   100] loss: 0.736 acc: 70.45 time: 3.53
[12,   200] loss: 0.734 acc: 70.64 time: 3.53
[12,   300] loss: 0.730 acc: 70.79 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 69.10 %
Average loss on the 10000 test images: 0.764
[13,   100] loss: 0.733 acc: 70.97 time: 3.54
[13,   200] loss: 0.726 acc: 70.98 time: 3.54
[13,   300] loss: 0.721 acc: 71.39 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 71.66 %
Average loss on the 10000 test images: 0.717
[14,   100] loss: 0.709 acc: 71.45 time: 3.55
[14,   200] loss: 0.704 acc: 71.71 time: 3.56
[14,   300] loss: 0.691 acc: 72.35 time: 3.61
TESTING:
Accuracy of the network on the 10000 test images: 72.68 %
Average loss on the 10000 test images: 0.684
[15,   100] loss: 0.699 acc: 71.54 time: 3.53
[15,   200] loss: 0.703 acc: 72.05 time: 3.53
[15,   300] loss: 0.686 acc: 72.81 time: 3.52
TESTING:
Accuracy of the network on the 10000 test images: 73.50 %
Average loss on the 10000 test images: 0.677
[16,   100] loss: 0.642 acc: 74.28 time: 3.56
[16,   200] loss: 0.637 acc: 74.96 time: 3.54
[16,   300] loss: 0.617 acc: 76.12 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 76.75 %
Average loss on the 10000 test images: 0.585
[17,   100] loss: 0.608 acc: 76.34 time: 3.54
[17,   200] loss: 0.595 acc: 76.59 time: 3.53
[17,   300] loss: 0.610 acc: 75.91 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 76.90 %
Average loss on the 10000 test images: 0.579
[18,   100] loss: 0.611 acc: 76.09 time: 3.53
[18,   200] loss: 0.602 acc: 76.42 time: 3.53
[18,   300] loss: 0.587 acc: 77.33 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 77.36 %
Average loss on the 10000 test images: 0.576
[19,   100] loss: 0.583 acc: 76.77 time: 3.55
[19,   200] loss: 0.594 acc: 76.91 time: 3.62
```

```
[19,   300] loss: 0.596 acc: 76.85 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 77.39 %
Average loss on the 10000 test images: 0.573
[20,   100] loss: 0.593 acc: 76.77 time: 3.53
[20,   200] loss: 0.580 acc: 77.15 time: 3.53
[20,   300] loss: 0.585 acc: 77.50 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 77.62 %
Average loss on the 10000 test images: 0.565
[21,   100] loss: 0.579 acc: 77.14 time: 3.54
[21,   200] loss: 0.591 acc: 76.62 time: 3.54
[21,   300] loss: 0.580 acc: 77.63 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 77.18 %
Average loss on the 10000 test images: 0.572
[22,   100] loss: 0.578 acc: 77.53 time: 3.54
[22,   200] loss: 0.574 acc: 77.57 time: 3.52
[22,   300] loss: 0.580 acc: 77.20 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 77.62 %
Average loss on the 10000 test images: 0.564
[23,   100] loss: 0.570 acc: 77.81 time: 3.54
[23,   200] loss: 0.595 acc: 76.94 time: 3.53
[23,   300] loss: 0.553 acc: 78.44 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 77.92 %
Average loss on the 10000 test images: 0.555
[24,   100] loss: 0.571 acc: 77.77 time: 3.62
[24,   200] loss: 0.567 acc: 77.88 time: 3.52
[24,   300] loss: 0.579 acc: 77.32 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 78.18 %
Average loss on the 10000 test images: 0.552
[25,   100] loss: 0.584 acc: 77.41 time: 3.54
[25,   200] loss: 0.559 acc: 78.24 time: 3.53
[25,   300] loss: 0.557 acc: 78.20 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 78.22 %
Average loss on the 10000 test images: 0.553
[26,   100] loss: 0.568 acc: 77.89 time: 3.53
[26,   200] loss: 0.563 acc: 77.98 time: 3.53
[26,   300] loss: 0.572 acc: 77.65 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 78.16 %
Average loss on the 10000 test images: 0.555
[27,   100] loss: 0.558 acc: 78.00 time: 3.53
[27,   200] loss: 0.560 acc: 78.00 time: 3.52
[27,   300] loss: 0.555 acc: 78.30 time: 3.53
```

```
TESTING:
Accuracy of the network on the 10000 test images: 78.29 %
Average loss on the 10000 test images: 0.555
[28,   100] loss: 0.565 acc: 77.41 time: 3.53
[28,   200] loss: 0.549 acc: 78.53 time: 3.53
[28,   300] loss: 0.561 acc: 77.80 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 78.28 %
Average loss on the 10000 test images: 0.551
[29,   100] loss: 0.544 acc: 78.90 time: 3.54
[29,   200] loss: 0.557 acc: 78.45 time: 3.53
[29,   300] loss: 0.539 acc: 78.89 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 78.93 %
Average loss on the 10000 test images: 0.545
[30,   100] loss: 0.548 acc: 78.72 time: 3.54
[30,   200] loss: 0.543 acc: 79.13 time: 3.53
[30,   300] loss: 0.556 acc: 78.45 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 78.55 %
Average loss on the 10000 test images: 0.548
[31,   100] loss: 0.547 acc: 78.77 time: 3.55
[31,   200] loss: 0.549 acc: 78.72 time: 3.52
[31,   300] loss: 0.539 acc: 79.12 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 78.89 %
Average loss on the 10000 test images: 0.542
[32,   100] loss: 0.542 acc: 79.20 time: 3.54
[32,   200] loss: 0.547 acc: 78.71 time: 3.53
[32,   300] loss: 0.529 acc: 79.38 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 79.47 %
Average loss on the 10000 test images: 0.536
[33,   100] loss: 0.536 acc: 78.87 time: 3.53
[33,   200] loss: 0.540 acc: 78.77 time: 3.54
[33,   300] loss: 0.547 acc: 78.77 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 79.13 %
Average loss on the 10000 test images: 0.540
[34,   100] loss: 0.539 acc: 78.95 time: 3.54
[34,   200] loss: 0.550 acc: 78.43 time: 3.54
[34,   300] loss: 0.538 acc: 79.22 time: 3.54
TESTING:
Accuracy of the network on the 10000 test images: 78.67 %
Average loss on the 10000 test images: 0.536
[35,   100] loss: 0.530 acc: 79.41 time: 3.54
[35,   200] loss: 0.542 acc: 78.88 time: 3.53
[35,   300] loss: 0.536 acc: 79.07 time: 3.53
TESTING:
```

```
Accuracy of the network on the 10000 test images: 79.38 %
Average loss on the 10000 test images: 0.532
[36,   100] loss: 0.530 acc: 79.35 time: 3.54
[36,   200] loss: 0.537 acc: 79.05 time: 3.53
[36,   300] loss: 0.543 acc: 78.73 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 79.15 %
Average loss on the 10000 test images: 0.534
[37,   100] loss: 0.537 acc: 79.49 time: 3.54
[37,   200] loss: 0.543 acc: 78.64 time: 3.53
[37,   300] loss: 0.540 acc: 78.84 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 79.08 %
Average loss on the 10000 test images: 0.537
[38,   100] loss: 0.541 acc: 79.09 time: 3.55
[38,   200] loss: 0.540 acc: 79.03 time: 3.53
[38,   300] loss: 0.540 acc: 79.01 time: 3.53
TESTING:
Accuracy of the network on the 10000 test images: 79.07 %
Average loss on the 10000 test images: 0.531
[39,   100] loss: 0.528 acc: 79.73 time: 3.55
[39,   200] loss: 0.534 acc: 79.52 time: 3.54
[39,   300] loss: 0.542 acc: 78.52 time: 3.51
TESTING:
Accuracy of the network on the 10000 test images: 78.99 %
Average loss on the 10000 test images: 0.530
[40,   100] loss: 0.523 acc: 79.63 time: 3.51
[40,   200] loss: 0.531 acc: 79.06 time: 3.51
[40,   300] loss: 0.543 acc: 79.12 time: 3.51
TESTING:
Accuracy of the network on the 10000 test images: 79.36 %
Average loss on the 10000 test images: 0.537
[41,   100] loss: 0.538 acc: 79.26 time: 3.52
[41,   200] loss: 0.526 acc: 79.72 time: 3.50
[41,   300] loss: 0.541 acc: 78.88 time: 3.51
TESTING:
Accuracy of the network on the 10000 test images: 79.15 %
Average loss on the 10000 test images: 0.532
[42,   100] loss: 0.542 acc: 78.70 time: 3.51
[42,   200] loss: 0.530 acc: 79.24 time: 3.51
[42,   300] loss: 0.531 acc: 79.44 time: 3.50
TESTING:
Accuracy of the network on the 10000 test images: 79.26 %
Average loss on the 10000 test images: 0.528
[43,   100] loss: 0.548 acc: 78.79 time: 3.51
[43,   200] loss: 0.531 acc: 79.45 time: 3.51
[43,   300] loss: 0.528 acc: 79.39 time: 3.59
TESTING:
Accuracy of the network on the 10000 test images: 79.26 %
```

```
Average loss on the 10000 test images: 0.533
[44,   100] loss: 0.533 acc: 79.25 time: 3.51
[44,   200] loss: 0.533 acc: 79.34 time: 3.50
[44,   300] loss: 0.539 acc: 79.01 time: 3.51
TESTING:
Accuracy of the network on the 10000 test images: 78.87 %
Average loss on the 10000 test images: 0.534
[45,   100] loss: 0.536 acc: 78.89 time: 3.52
[45,   200] loss: 0.530 acc: 79.44 time: 3.50
[45,   300] loss: 0.534 acc: 79.02 time: 3.51
TESTING:
Accuracy of the network on the 10000 test images: 79.00 %
Average loss on the 10000 test images: 0.535
[46,   100] loss: 0.536 acc: 79.23 time: 3.52
[46,   200] loss: 0.532 acc: 79.01 time: 3.51
[46,   300] loss: 0.525 acc: 79.42 time: 3.50
TESTING:
Accuracy of the network on the 10000 test images: 79.16 %
Average loss on the 10000 test images: 0.534
[47,   100] loss: 0.525 acc: 79.18 time: 3.52
[47,   200] loss: 0.531 acc: 79.95 time: 3.50
[47,   300] loss: 0.541 acc: 78.54 time: 3.50
TESTING:
Accuracy of the network on the 10000 test images: 78.87 %
Average loss on the 10000 test images: 0.533
[48,   100] loss: 0.540 acc: 78.99 time: 3.51
[48,   200] loss: 0.530 acc: 79.76 time: 3.59
[48,   300] loss: 0.534 acc: 79.30 time: 3.50
TESTING:
Accuracy of the network on the 10000 test images: 79.42 %
Average loss on the 10000 test images: 0.529
[49,   100] loss: 0.546 acc: 78.80 time: 3.51
[49,   200] loss: 0.535 acc: 79.38 time: 3.50
[49,   300] loss: 0.537 acc: 79.53 time: 3.50
TESTING:
Accuracy of the network on the 10000 test images: 79.11 %
Average loss on the 10000 test images: 0.530
[50,   100] loss: 0.528 acc: 79.25 time: 3.51
[50,   200] loss: 0.536 acc: 79.16 time: 3.51
[50,   300] loss: 0.535 acc: 79.02 time: 3.50
TESTING:
Accuracy of the network on the 10000 test images: 79.36 %
Average loss on the 10000 test images: 0.529
Finished Training
Saving Model ...
Saved Model !
```

2.1 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
net = resnet18(num_classes=4)
net.load_state_dict(torch.load("./models/resnet18_rotation.pth")
["parameters"])
device = "mps"
net.to(device)
```

```
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)
)

# TODO: Freeze all previous layers; only keep the 'layer4' block and
'fc' layer trainable
for p in net.parameters() :
    p.requires_grad = False

for weights in net.layer4.parameters() :
    weights.requires_grad = True

net.fc = torch.nn.Linear(net.fc.in_features, 10, True, device)

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

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

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

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

```
[1,   100] loss: 1.822 acc: 35.46 time: 4.73
[1,   200] loss: 1.585 acc: 42.75 time: 4.54
[1,   300] loss: 1.518 acc: 44.53 time: 4.53
TESTING:
Accuracy of the network on the 10000 test images: 50.89 %
Average loss on the 10000 test images: 1.400
[2,   100] loss: 1.427 acc: 48.48 time: 4.52
[2,   200] loss: 1.419 acc: 48.62 time: 4.62
[2,   300] loss: 1.384 acc: 50.52 time: 4.53
```

```
TESTING:
Accuracy of the network on the 10000 test images: 53.34 %
Average loss on the 10000 test images: 1.324
[3,   100] loss: 1.368 acc: 49.88 time: 4.58
[3,   200] loss: 1.366 acc: 50.86 time: 4.54
[3,   300] loss: 1.353 acc: 51.06 time: 4.64
TESTING:
Accuracy of the network on the 10000 test images: 56.47 %
Average loss on the 10000 test images: 1.240
[4,   100] loss: 1.339 acc: 51.43 time: 4.60
[4,   200] loss: 1.343 acc: 51.63 time: 4.55
[4,   300] loss: 1.313 acc: 52.63 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 54.48 %
Average loss on the 10000 test images: 1.304
[5,   100] loss: 1.317 acc: 52.50 time: 4.65
[5,   200] loss: 1.313 acc: 52.43 time: 4.67
[5,   300] loss: 1.303 acc: 53.39 time: 4.65
TESTING:
Accuracy of the network on the 10000 test images: 59.05 %
Average loss on the 10000 test images: 1.182
[6,   100] loss: 1.292 acc: 53.50 time: 4.67
[6,   200] loss: 1.300 acc: 53.72 time: 4.76
[6,   300] loss: 1.281 acc: 53.46 time: 4.70
TESTING:
Accuracy of the network on the 10000 test images: 56.05 %
Average loss on the 10000 test images: 1.244
[7,   100] loss: 1.269 acc: 54.73 time: 4.83
[7,   200] loss: 1.271 acc: 54.44 time: 4.82
[7,   300] loss: 1.268 acc: 54.51 time: 4.77
TESTING:
Accuracy of the network on the 10000 test images: 58.72 %
Average loss on the 10000 test images: 1.171
[8,   100] loss: 1.233 acc: 55.62 time: 4.79
[8,   200] loss: 1.261 acc: 54.72 time: 4.79
[8,   300] loss: 1.269 acc: 54.36 time: 4.71
TESTING:
Accuracy of the network on the 10000 test images: 60.40 %
Average loss on the 10000 test images: 1.132
[9,   100] loss: 1.262 acc: 54.06 time: 4.83
[9,   200] loss: 1.241 acc: 55.26 time: 4.74
[9,   300] loss: 1.228 acc: 55.45 time: 4.69
TESTING:
Accuracy of the network on the 10000 test images: 59.06 %
Average loss on the 10000 test images: 1.160
[10,   100] loss: 1.221 acc: 56.23 time: 4.78
[10,   200] loss: 1.229 acc: 56.12 time: 5.05
[10,   300] loss: 1.243 acc: 55.73 time: 4.89
TESTING:
```

```
Accuracy of the network on the 10000 test images: 60.12 %
Average loss on the 10000 test images: 1.129
[11,   100] loss: 1.184 acc: 57.87 time: 4.73
[11,   200] loss: 1.172 acc: 57.92 time: 4.70
[11,   300] loss: 1.176 acc: 57.82 time: 4.70
TESTING:
Accuracy of the network on the 10000 test images: 61.92 %
Average loss on the 10000 test images: 1.062
[12,   100] loss: 1.168 acc: 57.71 time: 4.74
[12,   200] loss: 1.161 acc: 58.79 time: 4.67
[12,   300] loss: 1.143 acc: 58.90 time: 4.72
TESTING:
Accuracy of the network on the 10000 test images: 62.53 %
Average loss on the 10000 test images: 1.056
[13,   100] loss: 1.143 acc: 58.49 time: 4.72
[13,   200] loss: 1.150 acc: 58.99 time: 4.70
[13,   300] loss: 1.153 acc: 58.56 time: 4.72
TESTING:
Accuracy of the network on the 10000 test images: 62.50 %
Average loss on the 10000 test images: 1.055
[14,   100] loss: 1.156 acc: 58.78 time: 4.67
[14,   200] loss: 1.138 acc: 59.20 time: 4.70
[14,   300] loss: 1.150 acc: 59.06 time: 4.78
TESTING:
Accuracy of the network on the 10000 test images: 62.45 %
Average loss on the 10000 test images: 1.052
[15,   100] loss: 1.141 acc: 59.38 time: 4.76
[15,   200] loss: 1.132 acc: 59.30 time: 4.81
[15,   300] loss: 1.128 acc: 59.02 time: 4.93
TESTING:
Accuracy of the network on the 10000 test images: 62.51 %
Average loss on the 10000 test images: 1.056
[16,   100] loss: 1.117 acc: 59.97 time: 4.87
[16,   200] loss: 1.125 acc: 59.95 time: 4.84
[16,   300] loss: 1.123 acc: 59.70 time: 4.80
TESTING:
Accuracy of the network on the 10000 test images: 63.07 %
Average loss on the 10000 test images: 1.043
[17,   100] loss: 1.125 acc: 59.79 time: 4.76
[17,   200] loss: 1.130 acc: 59.55 time: 4.92
[17,   300] loss: 1.120 acc: 59.73 time: 4.39
TESTING:
Accuracy of the network on the 10000 test images: 63.29 %
Average loss on the 10000 test images: 1.035
[18,   100] loss: 1.123 acc: 59.41 time: 4.77
[18,   200] loss: 1.120 acc: 59.09 time: 4.81
[18,   300] loss: 1.122 acc: 59.30 time: 4.75
TESTING:
Accuracy of the network on the 10000 test images: 63.23 %
```

```
Average loss on the 10000 test images: 1.032
[19,   100] loss: 1.125 acc: 59.55 time: 4.78
[19,   200] loss: 1.138 acc: 59.31 time: 4.70
[19,   300] loss: 1.113 acc: 60.12 time: 4.70
TESTING:
Accuracy of the network on the 10000 test images: 63.06 %
Average loss on the 10000 test images: 1.039
[20,   100] loss: 1.104 acc: 59.74 time: 4.71
[20,   200] loss: 1.113 acc: 59.77 time: 4.67
[20,   300] loss: 1.110 acc: 60.01 time: 4.66
TESTING:
Accuracy of the network on the 10000 test images: 63.72 %
Average loss on the 10000 test images: 1.027
Finished Training
```

```python
print('Saving Model ...')
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()}, "./models/resnet18_task2.pth")
print('Saved Model !')
```

```
Saving Model ...
Saved Model !
```

2.2 Fine-tuning on the randomly initialized model

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

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

from torchvision.models import resnet18
net = resnet18(weights=None)
device = "mps"
net.to(device)
# TODO: Randomly initialize a ResNet18 model
```

```
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=1000, bias=True)
)
```

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

for weights in net.layer4.parameters() :
    weights.requires_grad = True

net.fc = torch.nn.Linear(net.fc.in_features, 10, True, device)

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

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

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

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

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

```
[1,   100] loss: 2.319 acc: 22.51 time: 4.57
[1,   200] loss: 2.010 acc: 26.96 time: 4.52
[1,   300] loss: 1.955 acc: 29.23 time: 4.50
TESTING:
Accuracy of the network on the 10000 test images: 34.26 %
Average loss on the 10000 test images: 1.806
[2,   100] loss: 1.902 acc: 30.81 time: 4.58
[2,   200] loss: 1.880 acc: 31.57 time: 4.52
[2,   300] loss: 1.849 acc: 32.22 time: 4.51
TESTING:
Accuracy of the network on the 10000 test images: 35.84 %
Average loss on the 10000 test images: 1.764
[3,   100] loss: 1.855 acc: 31.98 time: 4.52
[3,   200] loss: 1.830 acc: 33.41 time: 4.52
[3,   300] loss: 1.827 acc: 32.68 time: 4.51
TESTING:
Accuracy of the network on the 10000 test images: 37.51 %
Average loss on the 10000 test images: 1.720
[4,   100] loss: 1.822 acc: 33.75 time: 4.40
[4,   200] loss: 1.808 acc: 33.91 time: 4.44
[4,   300] loss: 1.823 acc: 33.54 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 37.57 %
Average loss on the 10000 test images: 1.724
[5,   100] loss: 1.807 acc: 34.16 time: 4.54
[5,   200] loss: 1.802 acc: 33.94 time: 4.57
[5,   300] loss: 1.791 acc: 34.51 time: 4.56
TESTING:
Accuracy of the network on the 10000 test images: 38.65 %
Average loss on the 10000 test images: 1.706
[6,   100] loss: 1.779 acc: 35.48 time: 4.65
[6,   200] loss: 1.789 acc: 34.97 time: 4.63
[6,   300] loss: 1.778 acc: 35.03 time: 4.59
TESTING:
Accuracy of the network on the 10000 test images: 38.86 %
```

```
Average loss on the 10000 test images: 1.693
[7,   100] loss: 1.780 acc: 34.67 time: 4.60
[7,   200] loss: 1.776 acc: 35.19 time: 4.56
[7,   300] loss: 1.764 acc: 36.27 time: 4.57
TESTING:
Accuracy of the network on the 10000 test images: 39.79 %
Average loss on the 10000 test images: 1.678
[8,   100] loss: 1.762 acc: 36.41 time: 4.67
[8,   200] loss: 1.767 acc: 36.16 time: 4.56
[8,   300] loss: 1.758 acc: 36.05 time: 4.59
TESTING:
Accuracy of the network on the 10000 test images: 39.34 %
Average loss on the 10000 test images: 1.694
[9,   100] loss: 1.748 acc: 35.91 time: 4.60
[9,   200] loss: 1.757 acc: 35.39 time: 4.62
[9,   300] loss: 1.757 acc: 36.45 time: 4.62
TESTING:
Accuracy of the network on the 10000 test images: 39.31 %
Average loss on the 10000 test images: 1.687
[10,   100] loss: 1.739 acc: 36.82 time: 4.70
[10,   200] loss: 1.758 acc: 35.72 time: 4.69
[10,   300] loss: 1.747 acc: 36.85 time: 4.64
TESTING:
Accuracy of the network on the 10000 test images: 40.74 %
Average loss on the 10000 test images: 1.658
[11,   100] loss: 1.731 acc: 37.15 time: 4.84
[11,   200] loss: 1.709 acc: 38.32 time: 4.90
[11,   300] loss: 1.709 acc: 38.14 time: 4.79
TESTING:
Accuracy of the network on the 10000 test images: 40.59 %
Average loss on the 10000 test images: 1.636
[12,   100] loss: 1.709 acc: 38.22 time: 4.90
[12,   200] loss: 1.693 acc: 38.70 time: 4.74
[12,   300] loss: 1.696 acc: 38.95 time: 4.71
TESTING:
Accuracy of the network on the 10000 test images: 41.07 %
Average loss on the 10000 test images: 1.625
[13,   100] loss: 1.688 acc: 38.96 time: 4.72
[13,   200] loss: 1.681 acc: 38.98 time: 4.69
[13,   300] loss: 1.684 acc: 39.40 time: 4.78
TESTING:
Accuracy of the network on the 10000 test images: 41.33 %
Average loss on the 10000 test images: 1.622
[14,   100] loss: 1.688 acc: 38.67 time: 4.75
[14,   200] loss: 1.691 acc: 38.49 time: 4.87
[14,   300] loss: 1.692 acc: 38.96 time: 4.69
TESTING:
Accuracy of the network on the 10000 test images: 41.74 %
Average loss on the 10000 test images: 1.613
```

```
[15,   100] loss: 1.669 acc: 39.51 time: 4.71
[15,   200] loss: 1.692 acc: 38.95 time: 4.76
[15,   300] loss: 1.685 acc: 39.38 time: 4.75
TESTING:
Accuracy of the network on the 10000 test images: 41.27 %
Average loss on the 10000 test images: 1.615
[16,   100] loss: 1.678 acc: 39.23 time: 4.70
[16,   200] loss: 1.681 acc: 39.13 time: 4.74
[16,   300] loss: 1.681 acc: 39.73 time: 4.72
TESTING:
Accuracy of the network on the 10000 test images: 41.87 %
Average loss on the 10000 test images: 1.601
[17,   100] loss: 1.681 acc: 39.66 time: 4.71
[17,   200] loss: 1.669 acc: 39.88 time: 4.65
[17,   300] loss: 1.681 acc: 38.77 time: 4.71
TESTING:
Accuracy of the network on the 10000 test images: 41.85 %
Average loss on the 10000 test images: 1.610
[18,   100] loss: 1.661 acc: 40.16 time: 4.73
[18,   200] loss: 1.657 acc: 39.77 time: 4.55
[18,   300] loss: 1.678 acc: 39.10 time: 4.52
TESTING:
Accuracy of the network on the 10000 test images: 42.31 %
Average loss on the 10000 test images: 1.601
[19,   100] loss: 1.675 acc: 39.30 time: 4.72
[19,   200] loss: 1.652 acc: 40.11 time: 4.75
[19,   300] loss: 1.672 acc: 39.35 time: 4.71
TESTING:
Accuracy of the network on the 10000 test images: 41.93 %
Average loss on the 10000 test images: 1.603
[20,   100] loss: 1.654 acc: 39.92 time: 4.72
[20,   200] loss: 1.653 acc: 40.39 time: 4.68
[20,   300] loss: 1.679 acc: 39.16 time: 4.72
TESTING:
Accuracy of the network on the 10000 test images: 42.31 %
Average loss on the 10000 test images: 1.598
Finished Training
```

3.1 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
net = resnet18(num_classes=4)
net.load_state_dict(torch.load("./models/resnet18_rotation.pth")
["parameters"])
net.fc = nn.Linear(512,10,True,device)
device = "mps"
net.to(device)
```

```
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 = optim.Adam(net.parameters(), lr=0.01)

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

```
[1,   100] loss: 2.124 acc: 23.70 time: 5.73
[1,   200] loss: 1.813 acc: 32.50 time: 5.51
[1,   300] loss: 1.694 acc: 37.63 time: 5.50
TESTING:
Accuracy of the network on the 10000 test images: 45.38 %
Average loss on the 10000 test images: 1.494
[2,   100] loss: 1.512 acc: 44.37 time: 5.53
```

```
[2,   200] loss: 1.456 acc: 47.11 time: 5.62
[2,   300] loss: 1.408 acc: 49.74 time: 5.55
TESTING:
Accuracy of the network on the 10000 test images: 54.40 %
Average loss on the 10000 test images: 1.258
[3,   100] loss: 1.284 acc: 54.49 time: 5.58
[3,   200] loss: 1.217 acc: 56.27 time: 5.59
[3,   300] loss: 1.206 acc: 56.76 time: 5.57
TESTING:
Accuracy of the network on the 10000 test images: 60.74 %
Average loss on the 10000 test images: 1.121
[4,   100] loss: 1.117 acc: 60.30 time: 5.60
[4,   200] loss: 1.100 acc: 60.99 time: 5.62
[4,   300] loss: 1.083 acc: 61.45 time: 5.58
TESTING:
Accuracy of the network on the 10000 test images: 63.73 %
Average loss on the 10000 test images: 1.053
[5,   100] loss: 1.027 acc: 63.41 time: 5.66
[5,   200] loss: 0.983 acc: 65.34 time: 5.63
[5,   300] loss: 0.998 acc: 64.60 time: 5.65
TESTING:
Accuracy of the network on the 10000 test images: 67.54 %
Average loss on the 10000 test images: 0.950
[6,   100] loss: 0.941 acc: 66.74 time: 5.63
[6,   200] loss: 0.930 acc: 67.32 time: 5.66
[6,   300] loss: 0.917 acc: 67.79 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 70.55 %
Average loss on the 10000 test images: 0.844
[7,   100] loss: 0.872 acc: 69.45 time: 5.63
[7,   200] loss: 0.853 acc: 70.34 time: 5.66
[7,   300] loss: 0.846 acc: 70.05 time: 5.61
TESTING:
Accuracy of the network on the 10000 test images: 72.51 %
Average loss on the 10000 test images: 0.785
[8,   100] loss: 0.802 acc: 72.14 time: 5.65
[8,   200] loss: 0.812 acc: 71.62 time: 5.67
[8,   300] loss: 0.810 acc: 71.80 time: 5.66
TESTING:
Accuracy of the network on the 10000 test images: 74.15 %
Average loss on the 10000 test images: 0.760
[9,   100] loss: 0.762 acc: 73.94 time: 5.61
[9,   200] loss: 0.768 acc: 73.25 time: 5.68
[9,   300] loss: 0.782 acc: 72.83 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 75.13 %
Average loss on the 10000 test images: 0.725
[10,   100] loss: 0.719 acc: 75.14 time: 5.64
[10,   200] loss: 0.739 acc: 74.81 time: 5.66
```

```
[10,   300] loss: 0.733 acc: 74.78 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 74.54 %
Average loss on the 10000 test images: 0.735
[11,   100] loss: 0.643 acc: 77.74 time: 5.62
[11,   200] loss: 0.612 acc: 78.91 time: 5.67
[11,   300] loss: 0.593 acc: 79.47 time: 5.62
TESTING:
Accuracy of the network on the 10000 test images: 79.23 %
Average loss on the 10000 test images: 0.601
[12,   100] loss: 0.558 acc: 80.43 time: 5.62
[12,   200] loss: 0.571 acc: 80.27 time: 5.62
[12,   300] loss: 0.569 acc: 80.23 time: 5.65
TESTING:
Accuracy of the network on the 10000 test images: 79.83 %
Average loss on the 10000 test images: 0.588
[13,   100] loss: 0.550 acc: 80.72 time: 5.63
[13,   200] loss: 0.554 acc: 80.47 time: 5.64
[13,   300] loss: 0.539 acc: 80.98 time: 5.61
TESTING:
Accuracy of the network on the 10000 test images: 80.08 %
Average loss on the 10000 test images: 0.581
[14,   100] loss: 0.534 acc: 81.51 time: 5.63
[14,   200] loss: 0.530 acc: 81.55 time: 5.65
[14,   300] loss: 0.541 acc: 81.18 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 80.01 %
Average loss on the 10000 test images: 0.583
[15,   100] loss: 0.520 acc: 81.75 time: 5.64
[15,   200] loss: 0.532 acc: 81.38 time: 5.66
[15,   300] loss: 0.523 acc: 81.90 time: 5.64
TESTING:
Accuracy of the network on the 10000 test images: 80.44 %
Average loss on the 10000 test images: 0.566
[16,   100] loss: 0.502 acc: 82.40 time: 5.63
[16,   200] loss: 0.508 acc: 82.04 time: 5.65
[16,   300] loss: 0.512 acc: 82.13 time: 5.64
TESTING:
Accuracy of the network on the 10000 test images: 80.41 %
Average loss on the 10000 test images: 0.571
[17,   100] loss: 0.498 acc: 82.55 time: 5.63
[17,   200] loss: 0.507 acc: 82.16 time: 5.66
[17,   300] loss: 0.502 acc: 82.23 time: 5.62
TESTING:
Accuracy of the network on the 10000 test images: 80.67 %
Average loss on the 10000 test images: 0.565
[18,   100] loss: 0.490 acc: 83.06 time: 5.66
[18,   200] loss: 0.501 acc: 82.90 time: 5.65
[18,   300] loss: 0.492 acc: 82.50 time: 5.66
```

```
TESTING:
Accuracy of the network on the 10000 test images: 80.76 %
Average loss on the 10000 test images: 0.573
[19,   100] loss: 0.491 acc: 82.98 time: 5.66
[19,   200] loss: 0.489 acc: 82.80 time: 5.63
[19,   300] loss: 0.480 acc: 83.30 time: 5.62
TESTING:
Accuracy of the network on the 10000 test images: 81.07 %
Average loss on the 10000 test images: 0.562
[20,   100] loss: 0.485 acc: 82.98 time: 5.64
[20,   200] loss: 0.469 acc: 83.45 time: 5.62
[20,   300] loss: 0.479 acc: 83.38 time: 5.62
TESTING:
Accuracy of the network on the 10000 test images: 80.68 %
Average loss on the 10000 test images: 0.558
Finished Training
```

```
print('Saving Model ...')
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()}, "./models/resnet18_task3.pth")
print('Saved Model !')
```

```
Saving Model ...
Saved Model !
```

3.2 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
net = resnet18(weights=None)
net.fc = nn.Linear(512,10,True,device)
device = "mps"
net.to(device)
```

```
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 = optim.Adam(net.parameters(), lr=0.01)

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

```
[1,   100] loss: 2.331 acc: 20.11 time: 5.55
[1,   200] loss: 1.946 acc: 28.20 time: 5.60
[1,   300] loss: 1.821 acc: 33.02 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 38.49 %
Average loss on the 10000 test images: 1.760
[2,   100] loss: 1.671 acc: 39.80 time: 5.63
[2,   200] loss: 1.590 acc: 41.15 time: 5.64
[2,   300] loss: 1.536 acc: 44.13 time: 5.63
TESTING:
Accuracy of the network on the 10000 test images: 48.66 %
Average loss on the 10000 test images: 1.375
[3,   100] loss: 1.438 acc: 47.77 time: 5.71
[3,   200] loss: 1.379 acc: 49.43 time: 5.66
[3,   300] loss: 1.359 acc: 50.65 time: 5.65
TESTING:
Accuracy of the network on the 10000 test images: 54.25 %
Average loss on the 10000 test images: 1.280
[4,   100] loss: 1.247 acc: 54.94 time: 5.77
[4,   200] loss: 1.221 acc: 56.02 time: 5.71
[4,   300] loss: 1.213 acc: 56.80 time: 5.66
TESTING:
Accuracy of the network on the 10000 test images: 61.00 %
```

```
Average loss on the 10000 test images: 1.149
[5,   100] loss: 1.149 acc: 59.43 time: 5.73
[5,   200] loss: 1.126 acc: 59.79 time: 5.65
[5,   300] loss: 1.103 acc: 60.47 time: 5.64
TESTING:
Accuracy of the network on the 10000 test images: 66.59 %
Average loss on the 10000 test images: 0.949
[6,   100] loss: 1.057 acc: 62.83 time: 5.68
[6,   200] loss: 1.043 acc: 63.34 time: 5.76
[6,   300] loss: 1.020 acc: 63.56 time: 5.75
TESTING:
Accuracy of the network on the 10000 test images: 67.42 %
Average loss on the 10000 test images: 0.934
[7,   100] loss: 0.970 acc: 65.98 time: 5.72
[7,   200] loss: 0.976 acc: 65.70 time: 5.75
[7,   300] loss: 0.961 acc: 66.30 time: 5.68
TESTING:
Accuracy of the network on the 10000 test images: 67.79 %
Average loss on the 10000 test images: 0.980
[8,   100] loss: 0.927 acc: 66.88 time: 5.70
[8,   200] loss: 0.898 acc: 69.19 time: 5.74
[8,   300] loss: 0.905 acc: 68.03 time: 5.69
TESTING:
Accuracy of the network on the 10000 test images: 71.87 %
Average loss on the 10000 test images: 0.819
[9,   100] loss: 0.858 acc: 70.07 time: 5.69
[9,   200] loss: 0.856 acc: 70.06 time: 5.73
[9,   300] loss: 0.869 acc: 69.46 time: 5.69
TESTING:
Accuracy of the network on the 10000 test images: 72.85 %
Average loss on the 10000 test images: 0.799
[10,   100] loss: 0.812 acc: 71.53 time: 5.68
[10,   200] loss: 0.806 acc: 71.98 time: 5.78
[10,   300] loss: 0.808 acc: 71.88 time: 5.84
TESTING:
Accuracy of the network on the 10000 test images: 73.23 %
Average loss on the 10000 test images: 0.792
[11,   100] loss: 0.701 acc: 75.30 time: 5.67
[11,   200] loss: 0.672 acc: 76.63 time: 5.68
[11,   300] loss: 0.648 acc: 77.18 time: 5.75
TESTING:
Accuracy of the network on the 10000 test images: 77.53 %
Average loss on the 10000 test images: 0.654
[12,   100] loss: 0.649 acc: 77.60 time: 5.74
[12,   200] loss: 0.631 acc: 77.71 time: 5.70
[12,   300] loss: 0.638 acc: 77.73 time: 5.76
TESTING:
Accuracy of the network on the 10000 test images: 78.14 %
Average loss on the 10000 test images: 0.632
```

```
[13,   100] loss: 0.606 acc: 78.73 time: 5.74
[13,   200] loss: 0.623 acc: 78.09 time: 5.73
[13,   300] loss: 0.607 acc: 78.75 time: 5.78
TESTING:
Accuracy of the network on the 10000 test images: 78.23 %
Average loss on the 10000 test images: 0.635
[14,   100] loss: 0.586 acc: 79.51 time: 5.69
[14,   200] loss: 0.606 acc: 78.96 time: 5.76
[14,   300] loss: 0.600 acc: 79.22 time: 5.73
TESTING:
Accuracy of the network on the 10000 test images: 78.68 %
Average loss on the 10000 test images: 0.616
[15,   100] loss: 0.577 acc: 80.11 time: 5.72
[15,   200] loss: 0.588 acc: 79.65 time: 5.76
[15,   300] loss: 0.585 acc: 79.70 time: 5.73
TESTING:
Accuracy of the network on the 10000 test images: 78.95 %
Average loss on the 10000 test images: 0.615
[16,   100] loss: 0.562 acc: 80.39 time: 5.73
[16,   200] loss: 0.568 acc: 80.19 time: 5.72
[16,   300] loss: 0.569 acc: 80.23 time: 5.71
TESTING:
Accuracy of the network on the 10000 test images: 79.00 %
Average loss on the 10000 test images: 0.599
[17,   100] loss: 0.548 acc: 80.64 time: 5.89
[17,   200] loss: 0.555 acc: 80.19 time: 5.70
[17,   300] loss: 0.572 acc: 80.20 time: 5.66
TESTING:
Accuracy of the network on the 10000 test images: 79.66 %
Average loss on the 10000 test images: 0.588
[18,   100] loss: 0.542 acc: 81.14 time: 5.60
[18,   200] loss: 0.552 acc: 81.07 time: 5.63
[18,   300] loss: 0.547 acc: 81.02 time: 5.76
TESTING:
Accuracy of the network on the 10000 test images: 79.34 %
Average loss on the 10000 test images: 0.589
[19,   100] loss: 0.528 acc: 81.14 time: 5.73
[19,   200] loss: 0.540 acc: 81.07 time: 5.67
[19,   300] loss: 0.540 acc: 81.13 time: 5.65
TESTING:
Accuracy of the network on the 10000 test images: 79.52 %
Average loss on the 10000 test images: 0.590
[20,   100] loss: 0.522 acc: 81.88 time: 5.78
[20,   200] loss: 0.524 acc: 81.78 time: 5.82
[20,   300] loss: 0.531 acc: 81.59 time: 5.80
TESTING:
Accuracy of the network on the 10000 test images: 80.10 %
Average loss on the 10000 test images: 0.581
Finished Training
```

4a : Trying a better model

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

from torchvision.models import resnet34
from torch.optim.lr_scheduler import CosineAnnealingLR


net = resnet34(num_classes=4)
net = net.to(device)

# TODO: Define criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(net.parameters(), lr=0.0001, weight_decay=1e-
6)
scheduler = CosineAnnealingLR(optimizer, T_max=30, eta_min=1e-6)

def train_custom(net, criterion, optimizer, num_epochs, scheduler,
task):

    for epoch in range(num_epochs):

        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):

            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)

            optimizer.zero_grad()
            outputs = net(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            predicted = torch.max(outputs,dim=1).indices

            print_freq = 100
            running_loss += loss.item()
            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()

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

        net.eval()
        run_test(net,testloader,criterion,task)
        scheduler.step()

    print('Finished Training')
```

```
checkpoint = torch.load("./models/resnet_custom_rotation.pth",map_location=torch.device('mps'))
net.load_state_dict(checkpoint['parameters'])
```

```
<All keys matched successfully>
```

```
net.eval()
run_test(net,testloader,criterion,"rotation")
net.train()
```

```
TESTING:
Accuracy of the network on the 10000 test images: 86.07 %
Average loss on the 10000 test images: 0.445
```

```
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)
    )
    (2): 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)
    )
    (2): 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)
    )
    (3): 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)
    )
    (2): 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)
    )
    (3): 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)
    )
    (4): 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)
    )
    (5): 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)
    )
    (2): 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)
)
```

```
train_custom(net, criterion, optimizer, num_epochs=40,
scheduler=scheduler, task='rotation')
```

```
[1,   100] loss: 0.263 acc: 90.00 time: 7.24
[1,   200] loss: 0.241 acc: 91.11 time: 7.19
[1,   300] loss: 0.258 acc: 90.31 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 86.47 %
Average loss on the 10000 test images: 0.423
[2,   100] loss: 0.252 acc: 90.46 time: 7.26
[2,   200] loss: 0.247 acc: 90.41 time: 7.25
[2,   300] loss: 0.254 acc: 90.66 time: 7.20
TESTING:
Accuracy of the network on the 10000 test images: 86.34 %
Average loss on the 10000 test images: 0.415
[3,   100] loss: 0.247 acc: 90.69 time: 7.26
[3,   200] loss: 0.251 acc: 90.49 time: 7.22
[3,   300] loss: 0.251 acc: 90.61 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 86.32 %
Average loss on the 10000 test images: 0.418
[4,   100] loss: 0.242 acc: 91.05 time: 7.26
[4,   200] loss: 0.246 acc: 90.99 time: 7.22
[4,   300] loss: 0.246 acc: 90.76 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 86.32 %
Average loss on the 10000 test images: 0.426
[5,   100] loss: 0.245 acc: 90.66 time: 7.22
[5,   200] loss: 0.250 acc: 90.44 time: 7.22
[5,   300] loss: 0.245 acc: 90.88 time: 7.27
TESTING:
Accuracy of the network on the 10000 test images: 86.11 %
Average loss on the 10000 test images: 0.419
[6,   100] loss: 0.247 acc: 90.63 time: 7.23
[6,   200] loss: 0.246 acc: 90.55 time: 7.24
[6,   300] loss: 0.244 acc: 90.65 time: 7.23
TESTING:
Accuracy of the network on the 10000 test images: 85.66 %
Average loss on the 10000 test images: 0.437
[7,   100] loss: 0.240 acc: 91.07 time: 7.31
[7,   200] loss: 0.245 acc: 90.99 time: 7.28
[7,   300] loss: 0.245 acc: 90.73 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 86.09 %
Average loss on the 10000 test images: 0.424
[8,   100] loss: 0.239 acc: 90.85 time: 7.25
[8,   200] loss: 0.242 acc: 91.14 time: 7.23
[8,   300] loss: 0.240 acc: 90.92 time: 7.22
TESTING:
Accuracy of the network on the 10000 test images: 86.06 %
```

```
Average loss on the 10000 test images: 0.439
[9,   100] loss: 0.245 acc: 90.90 time: 7.22
[9,   200] loss: 0.243 acc: 90.45 time: 7.28
[9,   300] loss: 0.245 acc: 90.88 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 86.14 %
Average loss on the 10000 test images: 0.435
[10,   100] loss: 0.238 acc: 90.82 time: 7.23
[10,   200] loss: 0.241 acc: 90.73 time: 7.24
[10,   300] loss: 0.234 acc: 91.40 time: 7.29
TESTING:
Accuracy of the network on the 10000 test images: 85.74 %
Average loss on the 10000 test images: 0.435
[11,   100] loss: 0.234 acc: 91.27 time: 7.25
[11,   200] loss: 0.236 acc: 91.16 time: 7.24
[11,   300] loss: 0.251 acc: 90.45 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.83 %
Average loss on the 10000 test images: 0.437
[12,   100] loss: 0.243 acc: 90.67 time: 7.24
[12,   200] loss: 0.231 acc: 91.25 time: 7.23
[12,   300] loss: 0.238 acc: 90.80 time: 7.23
TESTING:
Accuracy of the network on the 10000 test images: 86.00 %
Average loss on the 10000 test images: 0.434
[13,   100] loss: 0.246 acc: 90.62 time: 7.23
[13,   200] loss: 0.243 acc: 90.92 time: 7.25
[13,   300] loss: 0.244 acc: 90.83 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.59 %
Average loss on the 10000 test images: 0.438
[14,   100] loss: 0.232 acc: 91.55 time: 7.24
[14,   200] loss: 0.239 acc: 90.84 time: 7.21
[14,   300] loss: 0.234 acc: 91.07 time: 7.23
TESTING:
Accuracy of the network on the 10000 test images: 85.78 %
Average loss on the 10000 test images: 0.436
[15,   100] loss: 0.227 acc: 91.46 time: 7.25
[15,   200] loss: 0.239 acc: 91.05 time: 7.21
[15,   300] loss: 0.240 acc: 91.02 time: 7.22
TESTING:
Accuracy of the network on the 10000 test images: 85.87 %
Average loss on the 10000 test images: 0.429
[16,   100] loss: 0.230 acc: 91.49 time: 7.25
[16,   200] loss: 0.238 acc: 91.02 time: 7.25
[16,   300] loss: 0.235 acc: 91.24 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 85.81 %
Average loss on the 10000 test images: 0.440
```

```
[17,   100] loss: 0.228 acc: 91.40 time: 7.24
[17,   200] loss: 0.237 acc: 91.11 time: 7.31
[17,   300] loss: 0.232 acc: 91.30 time: 7.22
TESTING:
Accuracy of the network on the 10000 test images: 85.87 %
Average loss on the 10000 test images: 0.449
[18,   100] loss: 0.225 acc: 91.39 time: 7.25
[18,   200] loss: 0.234 acc: 91.15 time: 7.21
[18,   300] loss: 0.233 acc: 91.34 time: 7.12
TESTING:
Accuracy of the network on the 10000 test images: 85.89 %
Average loss on the 10000 test images: 0.448
[19,   100] loss: 0.247 acc: 90.81 time: 7.23
[19,   200] loss: 0.225 acc: 91.71 time: 6.99
[19,   300] loss: 0.234 acc: 91.10 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.70 %
Average loss on the 10000 test images: 0.442
[20,   100] loss: 0.236 acc: 91.11 time: 7.24
[20,   200] loss: 0.223 acc: 91.55 time: 7.18
[20,   300] loss: 0.232 acc: 91.21 time: 7.05
TESTING:
Accuracy of the network on the 10000 test images: 85.64 %
Average loss on the 10000 test images: 0.449
[21,   100] loss: 0.224 acc: 91.60 time: 7.12
[21,   200] loss: 0.239 acc: 91.03 time: 7.21
[21,   300] loss: 0.231 acc: 91.36 time: 7.15
TESTING:
Accuracy of the network on the 10000 test images: 86.15 %
Average loss on the 10000 test images: 0.440
[22,   100] loss: 0.230 acc: 91.41 time: 6.98
[22,   200] loss: 0.229 acc: 91.44 time: 7.11
[22,   300] loss: 0.237 acc: 91.20 time: 6.89
TESTING:
Accuracy of the network on the 10000 test images: 85.63 %
Average loss on the 10000 test images: 0.445
[23,   100] loss: 0.230 acc: 91.42 time: 7.00
[23,   200] loss: 0.232 acc: 91.39 time: 6.93
[23,   300] loss: 0.226 acc: 91.60 time: 7.11
TESTING:
Accuracy of the network on the 10000 test images: 86.07 %
Average loss on the 10000 test images: 0.434
[24,   100] loss: 0.227 acc: 91.51 time: 7.05
[24,   200] loss: 0.241 acc: 90.77 time: 7.10
[24,   300] loss: 0.229 acc: 91.41 time: 6.99
TESTING:
Accuracy of the network on the 10000 test images: 86.01 %
Average loss on the 10000 test images: 0.442
[25,   100] loss: 0.227 acc: 91.67 time: 7.26
```

```
[25,   200] loss: 0.225 acc: 91.77 time: 7.23
[25,   300] loss: 0.223 acc: 91.79 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.98 %
Average loss on the 10000 test images: 0.442
[26,   100] loss: 0.229 acc: 91.39 time: 7.23
[26,   200] loss: 0.233 acc: 91.10 time: 7.25
[26,   300] loss: 0.236 acc: 91.15 time: 7.30
TESTING:
Accuracy of the network on the 10000 test images: 86.02 %
Average loss on the 10000 test images: 0.445
[27,   100] loss: 0.228 acc: 91.31 time: 7.23
[27,   200] loss: 0.234 acc: 91.23 time: 7.30
[27,   300] loss: 0.232 acc: 91.22 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.94 %
Average loss on the 10000 test images: 0.446
[28,   100] loss: 0.228 acc: 91.70 time: 7.28
[28,   200] loss: 0.234 acc: 91.23 time: 7.23
[28,   300] loss: 0.223 acc: 91.75 time: 7.28
TESTING:
Accuracy of the network on the 10000 test images: 85.91 %
Average loss on the 10000 test images: 0.440
[29,   100] loss: 0.215 acc: 92.05 time: 7.24
[29,   200] loss: 0.222 acc: 91.53 time: 7.25
[29,   300] loss: 0.230 acc: 91.32 time: 7.26
TESTING:
Accuracy of the network on the 10000 test images: 86.08 %
Average loss on the 10000 test images: 0.440
[30,   100] loss: 0.223 acc: 91.72 time: 7.23
[30,   200] loss: 0.226 acc: 91.44 time: 7.22
[30,   300] loss: 0.224 acc: 91.37 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 85.36 %
Average loss on the 10000 test images: 0.455
[31,   100] loss: 0.233 acc: 91.21 time: 7.26
[31,   200] loss: 0.240 acc: 90.76 time: 7.26
[31,   300] loss: 0.228 acc: 91.48 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.95 %
Average loss on the 10000 test images: 0.435
[32,   100] loss: 0.229 acc: 91.42 time: 7.26
[32,   200] loss: 0.239 acc: 91.07 time: 7.23
[32,   300] loss: 0.234 acc: 91.39 time: 7.26
TESTING:
Accuracy of the network on the 10000 test images: 85.62 %
Average loss on the 10000 test images: 0.447
[33,   100] loss: 0.228 acc: 91.16 time: 7.26
[33,   200] loss: 0.218 acc: 92.02 time: 7.23
```

```
[33,   300] loss: 0.229 acc: 91.14 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.97 %
Average loss on the 10000 test images: 0.446
[34,   100] loss: 0.229 acc: 91.27 time: 7.26
[34,   200] loss: 0.230 acc: 91.58 time: 7.27
[34,   300] loss: 0.234 acc: 91.05 time: 7.23
TESTING:
Accuracy of the network on the 10000 test images: 86.04 %
Average loss on the 10000 test images: 0.433
[35,   100] loss: 0.224 acc: 91.48 time: 7.32
[35,   200] loss: 0.236 acc: 91.05 time: 7.25
[35,   300] loss: 0.230 acc: 91.30 time: 7.26
TESTING:
Accuracy of the network on the 10000 test images: 85.79 %
Average loss on the 10000 test images: 0.451
[36,   100] loss: 0.237 acc: 90.96 time: 7.25
[36,   200] loss: 0.232 acc: 91.44 time: 7.26
[36,   300] loss: 0.229 acc: 91.52 time: 7.22
TESTING:
Accuracy of the network on the 10000 test images: 85.75 %
Average loss on the 10000 test images: 0.439
[37,   100] loss: 0.227 acc: 91.43 time: 7.25
[37,   200] loss: 0.230 acc: 91.52 time: 7.32
[37,   300] loss: 0.226 acc: 91.41 time: 7.24
TESTING:
Accuracy of the network on the 10000 test images: 85.89 %
Average loss on the 10000 test images: 0.446
[38,   100] loss: 0.230 acc: 91.40 time: 7.30
[38,   200] loss: 0.233 acc: 91.12 time: 7.29
[38,   300] loss: 0.230 acc: 91.34 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 86.12 %
Average loss on the 10000 test images: 0.436
[39,   100] loss: 0.239 acc: 90.90 time: 7.25
[39,   200] loss: 0.222 acc: 91.55 time: 7.27
[39,   300] loss: 0.225 acc: 91.59 time: 7.25
TESTING:
Accuracy of the network on the 10000 test images: 85.91 %
Average loss on the 10000 test images: 0.445
[40,   100] loss: 0.219 acc: 91.91 time: 7.25
[40,   200] loss: 0.235 acc: 91.27 time: 7.26
[40,   300] loss: 0.236 acc: 91.11 time: 7.23
TESTING:
Accuracy of the network on the 10000 test images: 85.97 %
Average loss on the 10000 test images: 0.437
Finished Training

print('Saving Model ...')
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()},
"./models/resnet_custom_rotation.pth")
print('Saved Model !')
```

```
Saving Model ...
Saved Model !
```

4b Classification

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

from torchvision.models import efficientnet_v2_s
from torch.optim.lr_scheduler import CosineAnnealingLR


net = efficientnet_v2_s()
net.classifier[1] = torch.nn.Linear(net.classifier[1].in_features, 10)
net = net.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(net.parameters(), lr=0.0001,
weight_decay=0.5*1e-5)
scheduler = CosineAnnealingLR(optimizer, T_max=30, eta_min=1e-7)

checkpoint =
torch.load("./models/resnet_custom_classification.pth",map_location=to
rch.device('mps'))
net.load_state_dict(checkpoint['parameters'])

net.eval()
run_test(net,testloader,criterion,"classification")
net.train()
```

```
TESTING:
Accuracy of the network on the 10000 test images: 84.42 %
Average loss on the 10000 test images: 0.472
```

```
EfficientNet(
  (features): Sequential(
    (0): Conv2dNormActivation(
      (0): Conv2d(3, 24, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (1): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.0, mode=row)
      )
      (1): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.005, mode=row)
      )
    )
    (2): Sequential(
      (0): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(24, 96, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(96, 48, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.01, mode=row)
      )
      (1): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(192, 48, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.015000000000000003,
mode=row)
      )
      (2): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(192, 48, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.02, mode=row)
      )
      (3): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(192, 48, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.025, mode=row)
      )
```

```
    )
    (3): Sequential(
      (0): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(48, 192, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.030000000000000006,
mode=row)
      )
      (1): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.035, mode=row)
      )
      (2): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.04, mode=row)
      )
      (3): FusedMBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.045, mode=row)
      )
    )
    (4): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256, bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(256, 16, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(16, 256, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(128, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.05, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=512, bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(128, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.05500000000000001,
mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=512, bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(128, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.06000000000000001,
mode=row)
      )
      (3): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=512, bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(128, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.065, mode=row)
      )
      (4): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=512, bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(128, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.07, mode=row)
      )
      (5): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=512, bias=False)
            (1): BatchNorm2d(512, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(128, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.075, mode=row)
      )
    )
    (5): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(128, 768, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(768, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(768, 768, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=768, bias=False)
            (1): BatchNorm2d(768, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(768, 32, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(32, 768, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(768, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.08, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.085, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.09, mode=row)
      )
      (3): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.095, mode=row)
      )
      (4): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1, mode=row)
      )
      (5): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.10500000000000001,
mode=row)
      )
      (6): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.11000000000000001,
mode=row)
      )
      (7): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.11500000000000002,
mode=row)
      )
      (8): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(160, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.12000000000000002,
mode=row)
      )
    )
    (6): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=960, bias=False)
            (1): BatchNorm2d(960, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(960, 40, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(40, 960, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(960, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.125, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.13, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
```

```
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.135, mode=row)
      )
      (3): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.14, mode=row)
      )
      (4): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.14500000000000002,
mode=row)
      )
      (5): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
```

```
     )
     (1): Conv2dNormActivation(
      (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
      (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
      (2): SiLU(inplace=True)
     )
     (2): SqueezeExcitation(
      (avgpool): AdaptiveAvgPool2d(output_size=1)
      (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
      (activation): SiLU(inplace=True)
      (scale_activation): Sigmoid()
     )
     (3): Conv2dNormActivation(
      (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
     )
    )
    (stochastic_depth): StochasticDepth(p=0.15, mode=row)
   )
   (6): MBConv(
    (block): Sequential(
     (0): Conv2dNormActivation(
      (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
      (2): SiLU(inplace=True)
     )
     (1): Conv2dNormActivation(
      (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
      (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
      (2): SiLU(inplace=True)
     )
     (2): SqueezeExcitation(
      (avgpool): AdaptiveAvgPool2d(output_size=1)
      (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
      (activation): SiLU(inplace=True)
      (scale_activation): Sigmoid()
     )
     (3): Conv2dNormActivation(
      (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.155, mode=row)
      )
      (7): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.16, mode=row)
      )
      (8): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.165, mode=row)
      )
      (9): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.17, mode=row)
      )
      (10): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.175, mode=row)
      )
      (11): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.18, mode=row)
      )
      (12): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.185, mode=row)
      )
      (13): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.19, mode=row)
      )
      (14): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 1536, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1536, 1536, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1536, bias=False)
            (1): BatchNorm2d(1536, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1536, 64, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(64, 1536, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1536, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(256, eps=0.001, momentum=0.1,
affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.195, mode=row)
      )
    )
    (7): Conv2dNormActivation(
      (0): Conv2d(256, 1280, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (1): BatchNorm2d(1280, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      (2): SiLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=1)
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=True)
    (1): Linear(in_features=1280, out_features=10, bias=True)
  )
)
```

```
train_custom(net, criterion, optimizer, num_epochs=60,
scheduler=scheduler, task='classification')
```

```
[1,   100] loss: 1.419 acc: 49.14 time: 19.46
[1,   200] loss: 1.451 acc: 47.90 time: 19.22
[1,   300] loss: 1.483 acc: 47.45 time: 19.22
TESTING:
Accuracy of the network on the 10000 test images: 49.75 %
Average loss on the 10000 test images: 1.367
[2,   100] loss: 1.417 acc: 49.28 time: 19.56
[2,   200] loss: 1.358 acc: 52.05 time: 19.47
[2,   300] loss: 1.481 acc: 48.16 time: 19.21
TESTING:
Accuracy of the network on the 10000 test images: 41.86 %
Average loss on the 10000 test images: 1.861
[3,   100] loss: 1.610 acc: 40.84 time: 19.43
[3,   200] loss: 1.470 acc: 46.72 time: 19.28
```

```
[3,   300] loss: 1.393 acc: 49.63 time: 19.37
TESTING:
Accuracy of the network on the 10000 test images: 53.91 %
Average loss on the 10000 test images: 1.302
[4,   100] loss: 1.329 acc: 52.48 time: 19.39
[4,   200] loss: 1.380 acc: 50.88 time: 19.28
[4,   300] loss: 1.305 acc: 52.67 time: 19.31
TESTING:
Accuracy of the network on the 10000 test images: 56.51 %
Average loss on the 10000 test images: 1.276
[5,   100] loss: 1.241 acc: 55.52 time: 19.07
[5,   200] loss: 1.220 acc: 56.04 time: 19.33
[5,   300] loss: 1.215 acc: 56.16 time: 19.40
TESTING:
Accuracy of the network on the 10000 test images: 58.49 %
Average loss on the 10000 test images: 1.160
[6,   100] loss: 1.224 acc: 56.06 time: 19.38
[6,   200] loss: 1.214 acc: 56.64 time: 19.55
[6,   300] loss: 1.170 acc: 57.96 time: 19.57
TESTING:
Accuracy of the network on the 10000 test images: 60.88 %
Average loss on the 10000 test images: 1.081
[7,   100] loss: 1.136 acc: 59.05 time: 19.15
[7,   200] loss: 1.137 acc: 59.02 time: 19.30
[7,   300] loss: 1.130 acc: 58.88 time: 19.82
TESTING:
Accuracy of the network on the 10000 test images: 63.62 %
Average loss on the 10000 test images: 1.034
[8,   100] loss: 1.120 acc: 60.41 time: 19.58
[8,   200] loss: 1.096 acc: 60.85 time: 19.50
[8,   300] loss: 1.086 acc: 61.26 time: 19.47
TESTING:
Accuracy of the network on the 10000 test images: 64.65 %
Average loss on the 10000 test images: 0.983
[9,   100] loss: 1.045 acc: 62.90 time: 19.53
[9,   200] loss: 1.042 acc: 62.52 time: 19.50
[9,   300] loss: 1.055 acc: 62.45 time: 19.40
TESTING:
Accuracy of the network on the 10000 test images: 66.97 %
Average loss on the 10000 test images: 0.927
[10,   100] loss: 1.013 acc: 63.80 time: 19.51
[10,   200] loss: 1.010 acc: 63.50 time: 19.54
[10,   300] loss: 0.984 acc: 64.42 time: 19.28
TESTING:
Accuracy of the network on the 10000 test images: 68.22 %
Average loss on the 10000 test images: 0.892
[11,   100] loss: 0.971 acc: 65.42 time: 19.33
[11,   200] loss: 0.972 acc: 65.10 time: 19.45
[11,   300] loss: 0.964 acc: 65.44 time: 19.42
```

```
TESTING:
Accuracy of the network on the 10000 test images: 68.75 %
Average loss on the 10000 test images: 0.877
[12,   100] loss: 0.939 acc: 66.24 time: 19.47
[12,   200] loss: 0.935 acc: 66.26 time: 19.43
[12,   300] loss: 0.949 acc: 66.66 time: 19.43
TESTING:
Accuracy of the network on the 10000 test images: 70.75 %
Average loss on the 10000 test images: 0.835
[13,   100] loss: 0.901 acc: 67.56 time: 19.51
[13,   200] loss: 0.909 acc: 67.98 time: 19.60
[13,   300] loss: 0.897 acc: 67.77 time: 19.47
TESTING:
Accuracy of the network on the 10000 test images: 70.86 %
Average loss on the 10000 test images: 0.823
[14,   100] loss: 0.898 acc: 67.86 time: 19.65
[14,   200] loss: 0.892 acc: 68.36 time: 19.15
[14,   300] loss: 0.881 acc: 68.76 time: 19.15
TESTING:
Accuracy of the network on the 10000 test images: 72.06 %
Average loss on the 10000 test images: 0.800
[15,   100] loss: 0.847 acc: 69.58 time: 19.46
[15,   200] loss: 0.859 acc: 68.85 time: 19.49
[15,   300] loss: 0.846 acc: 70.00 time: 19.44
TESTING:
Accuracy of the network on the 10000 test images: 72.04 %
Average loss on the 10000 test images: 0.789
[16,   100] loss: 0.834 acc: 70.21 time: 19.52
[16,   200] loss: 0.843 acc: 70.31 time: 19.45
[16,   300] loss: 0.841 acc: 70.22 time: 19.42
TESTING:
Accuracy of the network on the 10000 test images: 72.78 %
Average loss on the 10000 test images: 0.778
[17,   100] loss: 0.820 acc: 70.52 time: 19.46
[17,   200] loss: 0.828 acc: 70.70 time: 19.36
[17,   300] loss: 0.825 acc: 70.63 time: 19.38
TESTING:
Accuracy of the network on the 10000 test images: 73.23 %
Average loss on the 10000 test images: 0.762
[18,   100] loss: 0.824 acc: 70.97 time: 19.23
[18,   200] loss: 0.811 acc: 70.73 time: 19.43
[18,   300] loss: 0.823 acc: 71.06 time: 19.49
TESTING:
Accuracy of the network on the 10000 test images: 73.50 %
Average loss on the 10000 test images: 0.760
[19,   100] loss: 0.816 acc: 70.96 time: 19.30
[19,   200] loss: 0.816 acc: 70.82 time: 19.30
[19,   300] loss: 0.802 acc: 71.64 time: 19.36
TESTING:
```

```
Accuracy of the network on the 10000 test images: 73.80 %
Average loss on the 10000 test images: 0.754
[20,   100] loss: 0.790 acc: 72.37 time: 19.41
[20,   200] loss: 0.808 acc: 70.83 time: 19.44
[20,   300] loss: 0.807 acc: 71.65 time: 19.40
TESTING:
Accuracy of the network on the 10000 test images: 73.51 %
Average loss on the 10000 test images: 0.755
[21,   100] loss: 0.812 acc: 71.09 time: 19.21
[21,   200] loss: 0.804 acc: 71.24 time: 19.13
[21,   300] loss: 0.804 acc: 71.45 time: 19.11
TESTING:
Accuracy of the network on the 10000 test images: 73.56 %
Average loss on the 10000 test images: 0.756
[22,   100] loss: 0.792 acc: 71.50 time: 19.15
[22,   200] loss: 0.800 acc: 71.45 time: 19.04
[22,   300] loss: 0.815 acc: 70.95 time: 19.09
TESTING:
Accuracy of the network on the 10000 test images: 73.36 %
Average loss on the 10000 test images: 0.757
[23,   100] loss: 0.800 acc: 71.29 time: 19.14
[23,   200] loss: 0.822 acc: 70.34 time: 19.09
[23,   300] loss: 0.813 acc: 71.52 time: 19.06
TESTING:
Accuracy of the network on the 10000 test images: 73.42 %
Average loss on the 10000 test images: 0.754
[24,   100] loss: 0.797 acc: 71.91 time: 19.17
[24,   200] loss: 0.813 acc: 70.73 time: 19.28
[24,   300] loss: 0.800 acc: 71.68 time: 19.10
TESTING:
Accuracy of the network on the 10000 test images: 73.61 %
Average loss on the 10000 test images: 0.753
[25,   100] loss: 0.804 acc: 71.52 time: 19.51
[25,   200] loss: 0.805 acc: 71.20 time: 19.80
[25,   300] loss: 0.804 acc: 71.31 time: 19.38
TESTING:
Accuracy of the network on the 10000 test images: 73.66 %
Average loss on the 10000 test images: 0.754
[26,   100] loss: 0.794 acc: 71.51 time: 19.53
[26,   200] loss: 0.802 acc: 71.47 time: 19.26
[26,   300] loss: 0.815 acc: 70.99 time: 19.18
TESTING:
Accuracy of the network on the 10000 test images: 73.95 %
Average loss on the 10000 test images: 0.748
[27,   100] loss: 0.792 acc: 71.77 time: 19.23
[27,   200] loss: 0.807 acc: 71.03 time: 19.22
[27,   300] loss: 0.817 acc: 71.08 time: 19.21
TESTING:
Accuracy of the network on the 10000 test images: 74.07 %
```

```
Average loss on the 10000 test images: 0.748
[28,   100] loss: 0.810 acc: 70.98 time: 19.29
[28,   200] loss: 0.794 acc: 71.62 time: 19.28
[28,   300] loss: 0.798 acc: 71.75 time: 19.19
TESTING:
Accuracy of the network on the 10000 test images: 73.67 %
Average loss on the 10000 test images: 0.745
[29,   100] loss: 0.794 acc: 71.74 time: 19.31
[29,   200] loss: 0.802 acc: 71.57 time: 19.20
[29,   300] loss: 0.798 acc: 71.75 time: 19.26
TESTING:
Accuracy of the network on the 10000 test images: 73.75 %
Average loss on the 10000 test images: 0.747
[30,   100] loss: 0.797 acc: 71.66 time: 19.24
[30,   200] loss: 0.795 acc: 71.87 time: 19.26
[30,   300] loss: 0.801 acc: 71.63 time: 19.17
TESTING:
Accuracy of the network on the 10000 test images: 74.16 %
Average loss on the 10000 test images: 0.748
[31,   100] loss: 0.809 acc: 71.20 time: 19.28
[31,   200] loss: 0.793 acc: 71.84 time: 19.23
[31,   300] loss: 0.805 acc: 71.65 time: 19.25
TESTING:
Accuracy of the network on the 10000 test images: 74.10 %
Average loss on the 10000 test images: 0.745
[32,   100] loss: 0.769 acc: 72.58 time: 19.24
[32,   200] loss: 0.781 acc: 72.23 time: 19.19
[32,   300] loss: 0.804 acc: 71.58 time: 19.20
TESTING:
Accuracy of the network on the 10000 test images: 74.84 %
Average loss on the 10000 test images: 0.724
[33,   100] loss: 0.788 acc: 72.47 time: 19.30
[33,   200] loss: 0.781 acc: 72.81 time: 19.22
[33,   300] loss: 0.776 acc: 72.12 time: 19.02
TESTING:
Accuracy of the network on the 10000 test images: 75.65 %
Average loss on the 10000 test images: 0.719
[34,   100] loss: 0.755 acc: 73.25 time: 19.26
[34,   200] loss: 0.810 acc: 71.52 time: 19.17
[34,   300] loss: 0.792 acc: 72.14 time: 19.08
TESTING:
Accuracy of the network on the 10000 test images: 74.88 %
Average loss on the 10000 test images: 0.731
[35,   100] loss: 0.789 acc: 72.18 time: 19.24
[35,   200] loss: 0.789 acc: 72.81 time: 19.21
[35,   300] loss: 0.756 acc: 73.40 time: 19.08
TESTING:
Accuracy of the network on the 10000 test images: 75.48 %
Average loss on the 10000 test images: 0.713
```

```
[36,   100] loss: 0.777 acc: 72.66 time: 18.81
[36,   200] loss: 0.773 acc: 72.62 time: 18.82
[36,   300] loss: 0.757 acc: 74.09 time: 18.79
TESTING:
Accuracy of the network on the 10000 test images: 75.24 %
Average loss on the 10000 test images: 0.720
[37,   100] loss: 0.745 acc: 73.72 time: 18.90
[37,   200] loss: 0.775 acc: 72.84 time: 18.82
[37,   300] loss: 0.751 acc: 73.83 time: 18.85
TESTING:
Accuracy of the network on the 10000 test images: 76.67 %
Average loss on the 10000 test images: 0.692
[38,   100] loss: 0.732 acc: 74.16 time: 18.88
[38,   200] loss: 0.754 acc: 73.85 time: 18.87
[38,   300] loss: 0.770 acc: 73.38 time: 18.89
TESTING:
Accuracy of the network on the 10000 test images: 71.72 %
Average loss on the 10000 test images: 0.845
[39,   100] loss: 1.000 acc: 65.53 time: 18.86
[39,   200] loss: 0.886 acc: 69.38 time: 18.86
[39,   300] loss: 0.816 acc: 71.83 time: 18.84
TESTING:
Accuracy of the network on the 10000 test images: 74.63 %
Average loss on the 10000 test images: 0.733
[40,   100] loss: 0.777 acc: 72.59 time: 19.29
[40,   200] loss: 0.881 acc: 69.64 time: 19.22
[40,   300] loss: 0.796 acc: 72.23 time: 19.22
TESTING:
Accuracy of the network on the 10000 test images: 73.62 %
Average loss on the 10000 test images: 0.763
[41,   100] loss: 0.874 acc: 69.85 time: 19.21
[41,   200] loss: 0.849 acc: 70.54 time: 19.20
[41,   300] loss: 0.822 acc: 71.16 time: 19.19
TESTING:
Accuracy of the network on the 10000 test images: 75.61 %
Average loss on the 10000 test images: 0.705
[42,   100] loss: 0.717 acc: 75.36 time: 19.25
[42,   200] loss: 0.706 acc: 75.62 time: 19.21
[42,   300] loss: 0.816 acc: 72.16 time: 19.21
TESTING:
Accuracy of the network on the 10000 test images: 76.17 %
Average loss on the 10000 test images: 0.688
[43,   100] loss: 0.863 acc: 70.37 time: 19.28
[43,   200] loss: 0.907 acc: 69.02 time: 19.23
[43,   300] loss: 0.739 acc: 74.58 time: 19.17
TESTING:
Accuracy of the network on the 10000 test images: 76.55 %
Average loss on the 10000 test images: 0.676
[44,   100] loss: 0.727 acc: 74.41 time: 19.22
```

```
[44,   200] loss: 0.761 acc: 73.45 time: 19.20
[44,   300] loss: 0.718 acc: 75.11 time: 19.17
TESTING:
Accuracy of the network on the 10000 test images: 77.77 %
Average loss on the 10000 test images: 0.647
[45,   100] loss: 0.680 acc: 76.36 time: 19.24
[45,   200] loss: 0.661 acc: 76.86 time: 19.14
[45,   300] loss: 0.682 acc: 75.95 time: 19.17
TESTING:
Accuracy of the network on the 10000 test images: 54.07 %
Average loss on the 10000 test images: 1.360
[46,   100] loss: 1.204 acc: 58.02 time: 19.27
[46,   200] loss: 1.181 acc: 60.45 time: 19.22
[46,   300] loss: 1.271 acc: 55.47 time: 19.21
TESTING:
Accuracy of the network on the 10000 test images: 71.08 %
Average loss on the 10000 test images: 0.835
[47,   100] loss: 0.986 acc: 65.84 time: 19.01
[47,   200] loss: 0.953 acc: 67.28 time: 18.73
[47,   300] loss: 0.830 acc: 70.84 time: 18.68
TESTING:
Accuracy of the network on the 10000 test images: 75.49 %
Average loss on the 10000 test images: 0.706
[48,   100] loss: 0.782 acc: 72.98 time: 18.75
[48,   200] loss: 0.767 acc: 74.00 time: 18.62
[48,   300] loss: 0.760 acc: 74.39 time: 18.65
TESTING:
Accuracy of the network on the 10000 test images: 77.74 %
Average loss on the 10000 test images: 0.638
[49,   100] loss: 0.678 acc: 76.65 time: 18.71
[49,   200] loss: 0.687 acc: 76.02 time: 18.72
[49,   300] loss: 0.707 acc: 75.27 time: 18.65
TESTING:
Accuracy of the network on the 10000 test images: 78.66 %
Average loss on the 10000 test images: 0.616
[50,   100] loss: 0.637 acc: 77.57 time: 18.73
[50,   200] loss: 0.678 acc: 76.49 time: 18.61
[50,   300] loss: 0.670 acc: 76.97 time: 18.69
TESTING:
Accuracy of the network on the 10000 test images: 79.04 %
Average loss on the 10000 test images: 0.606
[51,   100] loss: 0.607 acc: 78.97 time: 18.68
[51,   200] loss: 0.641 acc: 77.77 time: 18.73
[51,   300] loss: 0.626 acc: 78.19 time: 18.61
TESTING:
Accuracy of the network on the 10000 test images: 78.63 %
Average loss on the 10000 test images: 0.620
[52,   100] loss: 0.606 acc: 78.94 time: 18.84
[52,   200] loss: 0.607 acc: 79.01 time: 18.69
```

```
[52,   300] loss: 0.602 acc: 78.91 time: 18.73
TESTING:
Accuracy of the network on the 10000 test images: 79.94 %
Average loss on the 10000 test images: 0.590
[53,   100] loss: 0.566 acc: 80.31 time: 18.72
[53,   200] loss: 0.595 acc: 79.20 time: 18.75
[53,   300] loss: 0.613 acc: 78.77 time: 18.62
TESTING:
Accuracy of the network on the 10000 test images: 79.71 %
Average loss on the 10000 test images: 0.587
[54,   100] loss: 0.584 acc: 80.21 time: 18.69
[54,   200] loss: 0.588 acc: 79.85 time: 18.69
[54,   300] loss: 0.570 acc: 80.19 time: 18.69
TESTING:
Accuracy of the network on the 10000 test images: 80.36 %
Average loss on the 10000 test images: 0.579
[55,   100] loss: 0.588 acc: 79.52 time: 18.80
[55,   200] loss: 0.636 acc: 77.97 time: 18.63
[55,   300] loss: 1.370 acc: 55.30 time: 18.67
TESTING:
Accuracy of the network on the 10000 test images: 72.56 %
Average loss on the 10000 test images: 0.793
[56,   100] loss: 0.826 acc: 71.25 time: 18.66
[56,   200] loss: 0.971 acc: 67.23 time: 18.76
[56,   300] loss: 0.964 acc: 67.06 time: 18.70
TESTING:
Accuracy of the network on the 10000 test images: 76.76 %
Average loss on the 10000 test images: 0.674
[57,   100] loss: 0.758 acc: 74.20 time: 18.75
[57,   200] loss: 0.713 acc: 75.84 time: 18.67
[57,   300] loss: 0.670 acc: 76.68 time: 18.71
TESTING:
Accuracy of the network on the 10000 test images: 80.52 %
Average loss on the 10000 test images: 0.573
[58,   100] loss: 0.610 acc: 79.00 time: 18.74
[58,   200] loss: 0.612 acc: 78.89 time: 18.73
[58,   300] loss: 0.606 acc: 78.84 time: 18.70
TESTING:
Accuracy of the network on the 10000 test images: 81.48 %
Average loss on the 10000 test images: 0.542
[59,   100] loss: 0.590 acc: 79.45 time: 18.80
[59,   200] loss: 0.559 acc: 80.69 time: 18.69
[59,   300] loss: 0.550 acc: 80.70 time: 18.67
TESTING:
Accuracy of the network on the 10000 test images: 79.54 %
Average loss on the 10000 test images: 0.590
[60,   100] loss: 0.566 acc: 80.52 time: 18.67
[60,   200] loss: 0.554 acc: 81.21 time: 18.66
[60,   300] loss: 0.597 acc: 79.50 time: 18.68
TESTING:
```

```
Accuracy of the network on the 10000 test images: 79.66 %
Average loss on the 10000 test images: 0.593
Finished Training
```

```python
train_custom(net, criterion, optimizer, num_epochs=40,
scheduler=scheduler, task='classification')
```

```
[1,   100] loss: 0.581 acc: 79.92 time: 19.18
[1,   200] loss: 0.556 acc: 80.44 time: 19.04
[1,   300] loss: 0.546 acc: 81.36 time: 19.17
TESTING:
Accuracy of the network on the 10000 test images: 81.53 %
Average loss on the 10000 test images: 0.540
[2,   100] loss: 0.516 acc: 82.16 time: 19.15
[2,   200] loss: 0.505 acc: 82.44 time: 19.18
[2,   300] loss: 0.506 acc: 82.65 time: 19.20
TESTING:
Accuracy of the network on the 10000 test images: 82.53 %
Average loss on the 10000 test images: 0.518
[3,   100] loss: 0.500 acc: 82.52 time: 19.26
[3,   200] loss: 0.480 acc: 83.34 time: 19.25
[3,   300] loss: 0.495 acc: 82.74 time: 19.21
TESTING:
Accuracy of the network on the 10000 test images: 82.79 %
Average loss on the 10000 test images: 0.512
[4,   100] loss: 0.480 acc: 83.36 time: 19.24
[4,   200] loss: 0.471 acc: 83.91 time: 19.32
[4,   300] loss: 0.471 acc: 83.64 time: 19.28
TESTING:
Accuracy of the network on the 10000 test images: 83.09 %
Average loss on the 10000 test images: 0.502
[5,   100] loss: 0.456 acc: 84.09 time: 19.35
[5,   200] loss: 0.473 acc: 83.63 time: 19.29
[5,   300] loss: 0.470 acc: 83.54 time: 19.27
TESTING:
Accuracy of the network on the 10000 test images: 83.26 %
Average loss on the 10000 test images: 0.496
[6,   100] loss: 0.463 acc: 84.13 time: 19.24
[6,   200] loss: 0.457 acc: 84.02 time: 19.27
[6,   300] loss: 0.448 acc: 84.18 time: 19.29
TESTING:
Accuracy of the network on the 10000 test images: 83.51 %
Average loss on the 10000 test images: 0.493
[7,   100] loss: 0.450 acc: 84.48 time: 19.31
[7,   200] loss: 0.446 acc: 84.48 time: 19.25
[7,   300] loss: 0.440 acc: 84.39 time: 19.26
TESTING:
Accuracy of the network on the 10000 test images: 83.37 %
Average loss on the 10000 test images: 0.493
[8,   100] loss: 0.428 acc: 84.98 time: 19.39
```

```
[8,   200] loss: 0.443 acc: 84.36 time: 19.27
[8,   300] loss: 0.426 acc: 85.21 time: 19.22
TESTING:
Accuracy of the network on the 10000 test images: 83.60 %
Average loss on the 10000 test images: 0.497
[9,   100] loss: 0.425 acc: 85.27 time: 19.28
[9,   200] loss: 0.425 acc: 85.47 time: 19.23
[9,   300] loss: 0.416 acc: 85.16 time: 19.28
TESTING:
Accuracy of the network on the 10000 test images: 84.02 %
Average loss on the 10000 test images: 0.483
[10,   100] loss: 0.414 acc: 85.30 time: 19.31
[10,   200] loss: 0.423 acc: 85.16 time: 19.36
[10,   300] loss: 0.425 acc: 85.16 time: 19.29
TESTING:
Accuracy of the network on the 10000 test images: 83.76 %
Average loss on the 10000 test images: 0.488
[11,   100] loss: 0.411 acc: 85.56 time: 19.36
[11,   200] loss: 0.408 acc: 85.83 time: 19.31
[11,   300] loss: 0.418 acc: 85.45 time: 19.27
TESTING:
Accuracy of the network on the 10000 test images: 83.84 %
Average loss on the 10000 test images: 0.484
[12,   100] loss: 0.407 acc: 85.76 time: 19.33
[12,   200] loss: 0.414 acc: 85.59 time: 19.33
[12,   300] loss: 0.405 acc: 86.18 time: 19.30
TESTING:
Accuracy of the network on the 10000 test images: 83.76 %
Average loss on the 10000 test images: 0.484
[13,   100] loss: 0.408 acc: 85.51 time: 19.27
[13,   200] loss: 0.404 acc: 85.79 time: 19.20
[13,   300] loss: 0.403 acc: 85.98 time: 19.24
TESTING:
Accuracy of the network on the 10000 test images: 83.93 %
Average loss on the 10000 test images: 0.487
[14,   100] loss: 0.398 acc: 86.25 time: 19.28
[14,   200] loss: 0.414 acc: 85.45 time: 19.10
[14,   300] loss: 0.409 acc: 85.52 time: 19.24
TESTING:
Accuracy of the network on the 10000 test images: 83.74 %
Average loss on the 10000 test images: 0.487
[15,   100] loss: 0.376 acc: 86.84 time: 18.82
[15,   200] loss: 0.404 acc: 85.93 time: 18.76
[15,   300] loss: 0.403 acc: 86.31 time: 18.75
TESTING:
Accuracy of the network on the 10000 test images: 83.83 %
Average loss on the 10000 test images: 0.479
[16,   100] loss: 0.380 acc: 86.74 time: 18.73
[16,   200] loss: 0.405 acc: 86.19 time: 18.83
```

```
[16,   300] loss: 0.390 acc: 86.23 time: 18.79
TESTING:
Accuracy of the network on the 10000 test images: 84.19 %
Average loss on the 10000 test images: 0.481
[17,   100] loss: 0.368 acc: 87.12 time: 18.87
[17,   200] loss: 0.400 acc: 86.16 time: 18.75
[17,   300] loss: 0.408 acc: 85.94 time: 18.77
TESTING:
Accuracy of the network on the 10000 test images: 84.23 %
Average loss on the 10000 test images: 0.478
[18,   100] loss: 0.387 acc: 86.70 time: 18.76
[18,   200] loss: 0.388 acc: 86.44 time: 18.77
[18,   300] loss: 0.390 acc: 86.20 time: 18.76
TESTING:
Accuracy of the network on the 10000 test images: 84.15 %
Average loss on the 10000 test images: 0.478
[19,   100] loss: 0.384 acc: 86.76 time: 18.82
[19,   200] loss: 0.382 acc: 86.70 time: 18.71
[19,   300] loss: 0.384 acc: 86.63 time: 19.27
TESTING:
Accuracy of the network on the 10000 test images: 84.26 %
Average loss on the 10000 test images: 0.478
[20,   100] loss: 0.394 acc: 86.38 time: 19.34
[20,   200] loss: 0.376 acc: 86.57 time: 19.32
[20,   300] loss: 0.387 acc: 86.07 time: 19.25
TESTING:
Accuracy of the network on the 10000 test images: 84.26 %
Average loss on the 10000 test images: 0.477
[21,   100] loss: 0.375 acc: 86.58 time: 19.40
[21,   200] loss: 0.369 acc: 87.27 time: 19.29
[21,   300] loss: 0.394 acc: 86.25 time: 19.24
TESTING:
Accuracy of the network on the 10000 test images: 84.43 %
Average loss on the 10000 test images: 0.476
[22,   100] loss: 0.382 acc: 86.90 time: 19.28
[22,   200] loss: 0.371 acc: 86.91 time: 19.29
[22,   300] loss: 0.388 acc: 86.36 time: 19.28
TESTING:
Accuracy of the network on the 10000 test images: 84.42 %
Average loss on the 10000 test images: 0.476
[23,   100] loss: 0.374 acc: 87.06 time: 19.39
[23,   200] loss: 0.376 acc: 86.95 time: 19.34
[23,   300] loss: 0.373 acc: 86.77 time: 19.27
TESTING:
Accuracy of the network on the 10000 test images: 84.31 %
Average loss on the 10000 test images: 0.479
[24,   100] loss: 0.369 acc: 87.08 time: 19.38
[24,   200] loss: 0.378 acc: 86.22 time: 19.38
[24,   300] loss: 0.374 acc: 86.92 time: 19.32
```

```
TESTING:
Accuracy of the network on the 10000 test images: 84.28 %
Average loss on the 10000 test images: 0.477
[25,   100] loss: 0.366 acc: 87.33 time: 19.36
[25,   200] loss: 0.375 acc: 86.87 time: 19.34
[25,   300] loss: 0.371 acc: 87.24 time: 19.32
TESTING:
Accuracy of the network on the 10000 test images: 84.26 %
Average loss on the 10000 test images: 0.478
[26,   100] loss: 0.364 acc: 87.40 time: 19.38
[26,   200] loss: 0.374 acc: 86.87 time: 19.29
[26,   300] loss: 0.369 acc: 87.05 time: 19.28
TESTING:
Accuracy of the network on the 10000 test images: 84.23 %
Average loss on the 10000 test images: 0.479
[27,   100] loss: 0.374 acc: 87.10 time: 19.31
[27,   200] loss: 0.374 acc: 86.81 time: 19.33
[27,   300] loss: 0.369 acc: 87.16 time: 19.22
TESTING:
Accuracy of the network on the 10000 test images: 84.27 %
Average loss on the 10000 test images: 0.474
[28,   100] loss: 0.367 acc: 87.16 time: 19.32
[28,   200] loss: 0.373 acc: 87.05 time: 19.25
[28,   300] loss: 0.374 acc: 86.75 time: 19.25
TESTING:
Accuracy of the network on the 10000 test images: 84.30 %
Average loss on the 10000 test images: 0.477
[29,   100] loss: 0.369 acc: 87.28 time: 19.29
[29,   200] loss: 0.378 acc: 86.80 time: 19.23
[29,   300] loss: 0.354 acc: 87.62 time: 19.24
TESTING:
Accuracy of the network on the 10000 test images: 84.29 %
Average loss on the 10000 test images: 0.477
[30,   100] loss: 0.362 acc: 87.30 time: 19.27
[30,   200] loss: 0.383 acc: 86.75 time: 19.19
[30,   300] loss: 0.365 acc: 87.45 time: 19.31
TESTING:
Accuracy of the network on the 10000 test images: 84.30 %
Average loss on the 10000 test images: 0.475
[31,   100] loss: 0.371 acc: 87.08 time: 19.02
[31,   200] loss: 0.376 acc: 86.83 time: 18.96
[31,   300] loss: 0.371 acc: 86.93 time: 18.76
TESTING:
Accuracy of the network on the 10000 test images: 84.23 %
Average loss on the 10000 test images: 0.476
[32,   100] loss: 0.379 acc: 86.65 time: 18.67
[32,   200] loss: 0.371 acc: 86.98 time: 18.75
[32,   300] loss: 0.354 acc: 87.75 time: 18.70
TESTING:
```

```
Accuracy of the network on the 10000 test images: 84.25 %
Average loss on the 10000 test images: 0.476
[33,   100] loss: 0.373 acc: 86.65 time: 18.78
[33,   200] loss: 0.382 acc: 86.34 time: 18.75
[33,   300] loss: 0.365 acc: 87.04 time: 18.75
TESTING:
Accuracy of the network on the 10000 test images: 84.34 %
Average loss on the 10000 test images: 0.477
[34,   100] loss: 0.370 acc: 87.06 time: 18.75
[34,   200] loss: 0.374 acc: 87.03 time: 18.77
[34,   300] loss: 0.356 acc: 87.48 time: 18.74
TESTING:
Accuracy of the network on the 10000 test images: 84.24 %
Average loss on the 10000 test images: 0.474
[35,   100] loss: 0.364 acc: 86.96 time: 18.77
[35,   200] loss: 0.375 acc: 86.96 time: 18.72
[35,   300] loss: 0.364 acc: 87.03 time: 18.72
TESTING:
Accuracy of the network on the 10000 test images: 84.35 %
Average loss on the 10000 test images: 0.474
[36,   100] loss: 0.366 acc: 87.38 time: 18.75
[36,   200] loss: 0.362 acc: 87.51 time: 18.75
[36,   300] loss: 0.371 acc: 87.21 time: 18.73
TESTING:
Accuracy of the network on the 10000 test images: 84.44 %
Average loss on the 10000 test images: 0.476
[37,   100] loss: 0.367 acc: 87.30 time: 18.82
[37,   200] loss: 0.362 acc: 87.37 time: 18.77
[37,   300] loss: 0.378 acc: 86.47 time: 18.69
TESTING:
Accuracy of the network on the 10000 test images: 84.41 %
Average loss on the 10000 test images: 0.474
[38,   100] loss: 0.372 acc: 86.91 time: 18.73
[38,   200] loss: 0.360 acc: 87.22 time: 18.71
[38,   300] loss: 0.373 acc: 87.27 time: 18.78
TESTING:
Accuracy of the network on the 10000 test images: 84.24 %
Average loss on the 10000 test images: 0.478
[39,   100] loss: 0.371 acc: 86.72 time: 18.75
[39,   200] loss: 0.382 acc: 86.85 time: 18.71
[39,   300] loss: 0.369 acc: 87.18 time: 18.68
TESTING:
Accuracy of the network on the 10000 test images: 84.51 %
Average loss on the 10000 test images: 0.473
[40,   100] loss: 0.373 acc: 86.73 time: 18.84
[40,   200] loss: 0.361 acc: 87.27 time: 18.73
[40,   300] loss: 0.375 acc: 86.68 time: 18.78
TESTING:
Accuracy of the network on the 10000 test images: 84.42 %
```

```
Average loss on the 10000 test images: 0.472
Finished Training
```

```
print('Saving Model ...')
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()},
"./models/resnet_custom_classification.pth")
print('Saved Model !')
```

```
Saving Model ...
Saved Model !
```

Extra Credit

Classification with rotation pre-training

```
device = 'mps'

import torch
from torch.utils.data import DataLoader, Subset
import numpy as np

num_samples_per_class = 5000
full_trainset = CIFAR10Rotation(root='./data', train=True,
download=True, transform=transform_train)

labels = [cls_label.item() for _, _, _, cls_label in full_trainset]

selected_indices = []
for cls in range(10):
    cls_indices = np.where(np.array(labels) == cls)[0]
    selected_indices.extend(np.random.choice(cls_indices,
num_samples_per_class, replace=False))

subset_trainset = Subset(full_trainset, selected_indices)
trainloader = DataLoader(subset_trainset, batch_size=batch_size,
shuffle=True, num_workers=0)

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

from torchvision.models import resnet34
from torch.optim.lr_scheduler import CosineAnnealingLR


net = resnet34(num_classes=4)

criterion = nn.CrossEntropyLoss()
```

```
checkpoint = torch.load("./models/resnet_custom_rotation.pth")
net.load_state_dict(checkpoint['parameters'])

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

optimizer = optim.AdamW(net.parameters(), lr=0.001, weight_decay=1e-5)
scheduler = CosineAnnealingLR(optimizer, T_max=30, eta_min=1e-6)

train_custom(net, criterion, optimizer, num_epochs=30,
scheduler=scheduler, task='classification')
```

```
[1,   100] loss: 1.837 acc: 31.74 time: 8.59
[1,   200] loss: 1.240 acc: 54.78 time: 8.61
[1,   300] loss: 1.074 acc: 61.77 time: 8.66
TESTING:
Accuracy of the network on the 10000 test images: 68.21 %
Average loss on the 10000 test images: 0.894
[2,   100] loss: 0.922 acc: 67.38 time: 8.78
[2,   200] loss: 0.879 acc: 69.03 time: 8.81
[2,   300] loss: 0.847 acc: 70.66 time: 8.79
TESTING:
Accuracy of the network on the 10000 test images: 74.82 %
Average loss on the 10000 test images: 0.723
[3,   100] loss: 0.783 acc: 73.36 time: 8.90
[3,   200] loss: 0.764 acc: 73.26 time: 8.89
[3,   300] loss: 0.765 acc: 73.64 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 77.30 %
Average loss on the 10000 test images: 0.670
[4,   100] loss: 0.692 acc: 76.33 time: 8.87
[4,   200] loss: 0.691 acc: 75.90 time: 8.88
[4,   300] loss: 0.701 acc: 75.62 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 78.27 %
Average loss on the 10000 test images: 0.645
[5,   100] loss: 0.625 acc: 78.06 time: 8.89
[5,   200] loss: 0.649 acc: 77.66 time: 8.91
[5,   300] loss: 0.644 acc: 77.90 time: 8.87
TESTING:
Accuracy of the network on the 10000 test images: 79.81 %
Average loss on the 10000 test images: 0.605
[6,   100] loss: 0.587 acc: 79.84 time: 8.90
[6,   200] loss: 0.592 acc: 79.41 time: 8.92
[6,   300] loss: 0.603 acc: 79.42 time: 8.70
TESTING:
Accuracy of the network on the 10000 test images: 80.23 %
Average loss on the 10000 test images: 0.587
[7,   100] loss: 0.560 acc: 80.75 time: 8.92
```

```
[7,   200] loss: 0.552 acc: 80.45 time: 8.97
[7,   300] loss: 0.565 acc: 80.23 time: 8.96
TESTING:
Accuracy of the network on the 10000 test images: 81.14 %
Average loss on the 10000 test images: 0.563
[8,   100] loss: 0.529 acc: 81.64 time: 8.93
[8,   200] loss: 0.539 acc: 81.20 time: 8.92
[8,   300] loss: 0.516 acc: 82.12 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 81.35 %
Average loss on the 10000 test images: 0.555
[9,   100] loss: 0.506 acc: 82.81 time: 8.89
[9,   200] loss: 0.494 acc: 82.71 time: 8.92
[9,   300] loss: 0.499 acc: 83.00 time: 8.92
TESTING:
Accuracy of the network on the 10000 test images: 81.19 %
Average loss on the 10000 test images: 0.557
[10,   100] loss: 0.472 acc: 83.50 time: 8.84
[10,   200] loss: 0.476 acc: 83.50 time: 8.90
[10,   300] loss: 0.496 acc: 82.81 time: 8.90
TESTING:
Accuracy of the network on the 10000 test images: 82.30 %
Average loss on the 10000 test images: 0.540
[11,   100] loss: 0.450 acc: 84.47 time: 8.94
[11,   200] loss: 0.465 acc: 83.67 time: 8.87
[11,   300] loss: 0.461 acc: 83.89 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 82.42 %
Average loss on the 10000 test images: 0.523
[12,   100] loss: 0.438 acc: 84.73 time: 8.87
[12,   200] loss: 0.439 acc: 84.76 time: 8.86
[12,   300] loss: 0.444 acc: 84.69 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 82.90 %
Average loss on the 10000 test images: 0.521
[13,   100] loss: 0.406 acc: 85.86 time: 8.93
[13,   200] loss: 0.409 acc: 85.94 time: 8.95
[13,   300] loss: 0.414 acc: 85.60 time: 8.89
TESTING:
Accuracy of the network on the 10000 test images: 82.55 %
Average loss on the 10000 test images: 0.528
[14,   100] loss: 0.382 acc: 86.84 time: 8.87
[14,   200] loss: 0.401 acc: 85.81 time: 8.88
[14,   300] loss: 0.396 acc: 85.93 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 83.13 %
Average loss on the 10000 test images: 0.511
[15,   100] loss: 0.375 acc: 87.20 time: 8.90
[15,   200] loss: 0.380 acc: 86.51 time: 8.91
```

```
[15,   300] loss: 0.375 acc: 86.80 time: 8.90
TESTING:
Accuracy of the network on the 10000 test images: 83.48 %
Average loss on the 10000 test images: 0.520
[16,   100] loss: 0.349 acc: 88.23 time: 8.91
[16,   200] loss: 0.374 acc: 86.92 time: 8.92
[16,   300] loss: 0.360 acc: 87.80 time: 8.97
TESTING:
Accuracy of the network on the 10000 test images: 83.57 %
Average loss on the 10000 test images: 0.511
[17,   100] loss: 0.343 acc: 87.88 time: 8.91
[17,   200] loss: 0.354 acc: 87.78 time: 8.95
[17,   300] loss: 0.352 acc: 87.77 time: 8.87
TESTING:
Accuracy of the network on the 10000 test images: 83.91 %
Average loss on the 10000 test images: 0.521
[18,   100] loss: 0.309 acc: 89.08 time: 8.92
[18,   200] loss: 0.334 acc: 88.34 time: 8.95
[18,   300] loss: 0.342 acc: 88.01 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 83.93 %
Average loss on the 10000 test images: 0.505
[19,   100] loss: 0.320 acc: 88.80 time: 8.93
[19,   200] loss: 0.321 acc: 88.80 time: 8.91
[19,   300] loss: 0.314 acc: 88.85 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 84.24 %
Average loss on the 10000 test images: 0.509
[20,   100] loss: 0.300 acc: 89.67 time: 8.87
[20,   200] loss: 0.305 acc: 89.30 time: 8.94
[20,   300] loss: 0.300 acc: 89.39 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 84.04 %
Average loss on the 10000 test images: 0.515
[21,   100] loss: 0.287 acc: 89.87 time: 8.91
[21,   200] loss: 0.298 acc: 89.41 time: 8.96
[21,   300] loss: 0.301 acc: 89.48 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 84.29 %
Average loss on the 10000 test images: 0.513
[22,   100] loss: 0.287 acc: 90.08 time: 8.98
[22,   200] loss: 0.276 acc: 90.51 time: 8.96
[22,   300] loss: 0.295 acc: 89.61 time: 8.97
TESTING:
Accuracy of the network on the 10000 test images: 84.40 %
Average loss on the 10000 test images: 0.509
[23,   100] loss: 0.275 acc: 90.41 time: 9.00
[23,   200] loss: 0.267 acc: 90.66 time: 8.94
[23,   300] loss: 0.282 acc: 90.10 time: 8.97
```

```
TESTING:
Accuracy of the network on the 10000 test images: 84.20 %
Average loss on the 10000 test images: 0.516
[24,   100] loss: 0.262 acc: 90.99 time: 8.91
[24,   200] loss: 0.273 acc: 90.23 time: 8.88
[24,   300] loss: 0.264 acc: 90.60 time: 8.90
TESTING:
Accuracy of the network on the 10000 test images: 84.32 %
Average loss on the 10000 test images: 0.511
[25,   100] loss: 0.251 acc: 91.17 time: 8.91
[25,   200] loss: 0.259 acc: 90.91 time: 8.93
[25,   300] loss: 0.262 acc: 90.92 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 84.32 %
Average loss on the 10000 test images: 0.521
[26,   100] loss: 0.255 acc: 90.99 time: 8.89
[26,   200] loss: 0.254 acc: 91.12 time: 9.17
[26,   300] loss: 0.256 acc: 91.15 time: 9.06
TESTING:
Accuracy of the network on the 10000 test images: 84.65 %
Average loss on the 10000 test images: 0.510
[27,   100] loss: 0.236 acc: 91.66 time: 9.00
[27,   200] loss: 0.247 acc: 91.25 time: 8.93
[27,   300] loss: 0.253 acc: 91.17 time: 8.96
TESTING:
Accuracy of the network on the 10000 test images: 84.40 %
Average loss on the 10000 test images: 0.515
[28,   100] loss: 0.247 acc: 91.16 time: 8.95
[28,   200] loss: 0.250 acc: 90.98 time: 8.97
[28,   300] loss: 0.254 acc: 90.89 time: 8.95
TESTING:
Accuracy of the network on the 10000 test images: 84.68 %
Average loss on the 10000 test images: 0.521
[29,   100] loss: 0.251 acc: 91.04 time: 8.91
[29,   200] loss: 0.236 acc: 91.74 time: 8.97
[29,   300] loss: 0.254 acc: 91.13 time: 9.02
TESTING:
Accuracy of the network on the 10000 test images: 84.70 %
Average loss on the 10000 test images: 0.519
[30,   100] loss: 0.246 acc: 91.43 time: 8.94
[30,   200] loss: 0.249 acc: 91.22 time: 8.89
[30,   300] loss: 0.241 acc: 91.59 time: 8.89
TESTING:
Accuracy of the network on the 10000 test images: 84.46 %
Average loss on the 10000 test images: 0.517
Finished Training
```

Classification without pre-training

```
net = resnet34(num_classes=10)
net = net.to(device)
optimizer = optim.AdamW(net.parameters(), lr=0.001, weight_decay=1e-5)
scheduler = CosineAnnealingLR(optimizer, T_max=30, eta_min=1e-6)

train_custom(net, criterion, optimizer, num_epochs=30,
scheduler=scheduler, task='classification')
```

```
[1,   100] loss: 1.978 acc: 27.52 time: 8.87
[1,   200] loss: 1.734 acc: 36.22 time: 8.93
[1,   300] loss: 1.630 acc: 39.88 time: 8.88
TESTING:
Accuracy of the network on the 10000 test images: 47.48 %
Average loss on the 10000 test images: 1.428
[2,   100] loss: 1.446 acc: 47.47 time: 8.88
[2,   200] loss: 1.404 acc: 49.35 time: 8.89
[2,   300] loss: 1.354 acc: 51.59 time: 8.89
TESTING:
Accuracy of the network on the 10000 test images: 55.39 %
Average loss on the 10000 test images: 1.278
[3,   100] loss: 1.237 acc: 55.62 time: 8.99
[3,   200] loss: 1.202 acc: 57.34 time: 8.87
[3,   300] loss: 1.201 acc: 57.23 time: 8.90
TESTING:
Accuracy of the network on the 10000 test images: 62.46 %
Average loss on the 10000 test images: 1.089
[4,   100] loss: 1.104 acc: 60.87 time: 8.92
[4,   200] loss: 1.105 acc: 61.40 time: 8.91
[4,   300] loss: 1.079 acc: 62.27 time: 8.89
TESTING:
Accuracy of the network on the 10000 test images: 63.28 %
Average loss on the 10000 test images: 1.066
[5,   100] loss: 0.998 acc: 64.91 time: 8.89
[5,   200] loss: 1.009 acc: 64.47 time: 8.89
[5,   300] loss: 1.006 acc: 64.48 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 67.95 %
Average loss on the 10000 test images: 0.942
[6,   100] loss: 0.940 acc: 66.92 time: 9.00
[6,   200] loss: 0.922 acc: 67.53 time: 8.92
[6,   300] loss: 0.921 acc: 67.64 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 70.60 %
Average loss on the 10000 test images: 0.859
[7,   100] loss: 0.876 acc: 69.37 time: 8.89
[7,   200] loss: 0.915 acc: 68.08 time: 8.93
[7,   300] loss: 0.864 acc: 69.93 time: 8.89
TESTING:
```

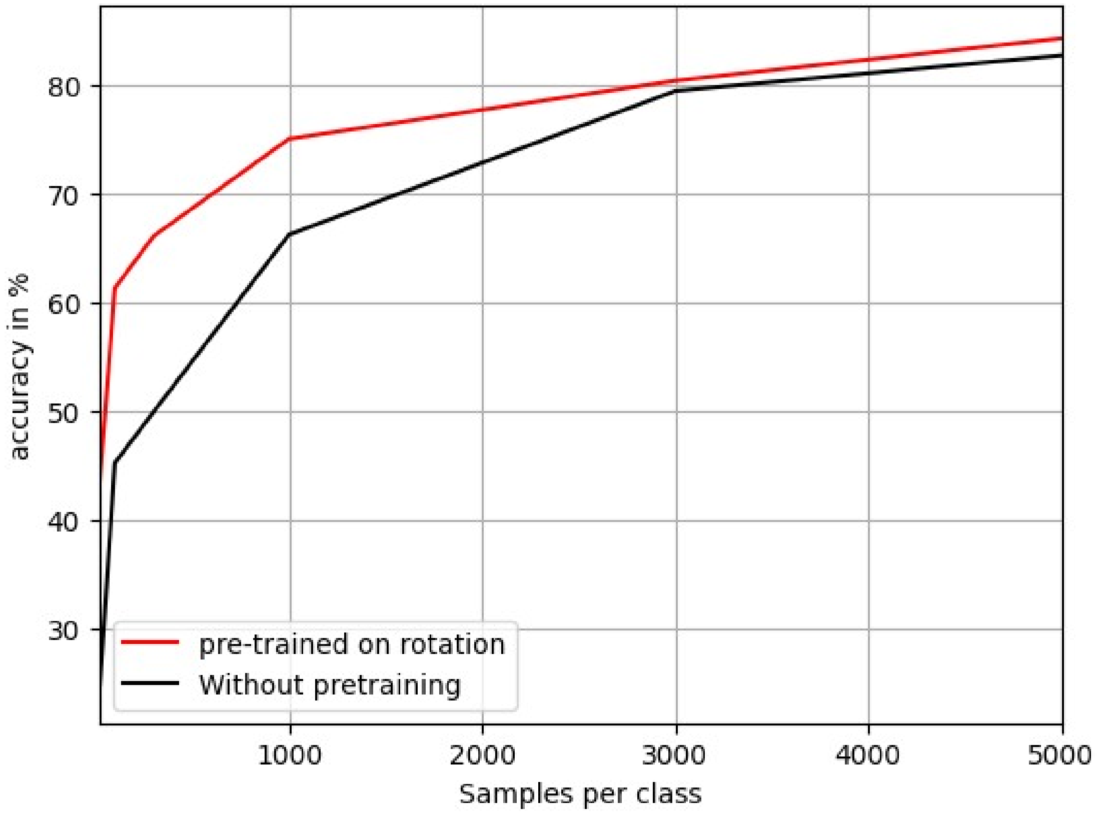
```
Accuracy of the network on the 10000 test images: 70.25 %
Average loss on the 10000 test images: 0.856
[8,   100] loss: 0.812 acc: 71.56 time: 8.95
[8,   200] loss: 0.863 acc: 70.41 time: 8.94
[8,   300] loss: 0.822 acc: 71.98 time: 8.91
TESTING:
Accuracy of the network on the 10000 test images: 73.70 %
Average loss on the 10000 test images: 0.773
[9,   100] loss: 0.751 acc: 73.79 time: 8.90
[9,   200] loss: 0.764 acc: 73.48 time: 8.97
[9,   300] loss: 0.760 acc: 73.70 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 74.41 %
Average loss on the 10000 test images: 0.737
[10,   100] loss: 0.717 acc: 75.07 time: 8.94
[10,   200] loss: 0.734 acc: 74.32 time: 8.89
[10,   300] loss: 0.736 acc: 74.45 time: 8.92
TESTING:
Accuracy of the network on the 10000 test images: 73.85 %
Average loss on the 10000 test images: 0.772
[11,   100] loss: 0.690 acc: 75.95 time: 8.89
[11,   200] loss: 0.698 acc: 75.27 time: 8.91
[11,   300] loss: 0.699 acc: 75.77 time: 8.73
TESTING:
Accuracy of the network on the 10000 test images: 75.64 %
Average loss on the 10000 test images: 0.720
[12,   100] loss: 0.717 acc: 75.14 time: 8.63
[12,   200] loss: 0.804 acc: 72.86 time: 8.62
[12,   300] loss: 0.717 acc: 75.29 time: 8.82
TESTING:
Accuracy of the network on the 10000 test images: 76.60 %
Average loss on the 10000 test images: 0.681
[13,   100] loss: 0.640 acc: 77.45 time: 8.57
[13,   200] loss: 0.633 acc: 77.93 time: 8.64
[13,   300] loss: 0.632 acc: 77.91 time: 8.77
TESTING:
Accuracy of the network on the 10000 test images: 79.10 %
Average loss on the 10000 test images: 0.613
[14,   100] loss: 0.584 acc: 79.54 time: 8.72
[14,   200] loss: 0.591 acc: 79.38 time: 8.59
[14,   300] loss: 0.611 acc: 78.41 time: 8.65
TESTING:
Accuracy of the network on the 10000 test images: 78.28 %
Average loss on the 10000 test images: 0.633
[15,   100] loss: 0.562 acc: 80.62 time: 8.70
[15,   200] loss: 0.558 acc: 80.55 time: 8.59
[15,   300] loss: 0.570 acc: 80.27 time: 8.64
TESTING:
Accuracy of the network on the 10000 test images: 79.59 %
```

```
Average loss on the 10000 test images: 0.596
[16,   100] loss: 0.522 acc: 81.74 time: 8.74
[16,   200] loss: 0.542 acc: 81.16 time: 8.61
[16,   300] loss: 0.547 acc: 81.09 time: 8.65
TESTING:
Accuracy of the network on the 10000 test images: 79.99 %
Average loss on the 10000 test images: 0.598
[17,   100] loss: 0.504 acc: 82.28 time: 8.72
[17,   200] loss: 0.501 acc: 82.75 time: 8.60
[17,   300] loss: 0.518 acc: 81.77 time: 8.67
TESTING:
Accuracy of the network on the 10000 test images: 80.49 %
Average loss on the 10000 test images: 0.585
[18,   100] loss: 0.481 acc: 83.15 time: 8.75
[18,   200] loss: 0.475 acc: 83.65 time: 8.58
[18,   300] loss: 0.501 acc: 82.19 time: 8.66
TESTING:
Accuracy of the network on the 10000 test images: 80.97 %
Average loss on the 10000 test images: 0.563
[19,   100] loss: 0.452 acc: 84.20 time: 8.84
[19,   200] loss: 0.454 acc: 83.93 time: 8.61
[19,   300] loss: 0.464 acc: 83.83 time: 8.63
TESTING:
Accuracy of the network on the 10000 test images: 81.22 %
Average loss on the 10000 test images: 0.568
[20,   100] loss: 0.428 acc: 84.97 time: 8.79
[20,   200] loss: 0.444 acc: 84.48 time: 8.70
[20,   300] loss: 0.445 acc: 84.48 time: 8.64
TESTING:
Accuracy of the network on the 10000 test images: 80.83 %
Average loss on the 10000 test images: 0.573
[21,   100] loss: 0.410 acc: 85.68 time: 8.74
[21,   200] loss: 0.415 acc: 85.23 time: 8.71
[21,   300] loss: 0.421 acc: 85.34 time: 8.61
TESTING:
Accuracy of the network on the 10000 test images: 81.97 %
Average loss on the 10000 test images: 0.549
[22,   100] loss: 0.395 acc: 86.04 time: 8.82
[22,   200] loss: 0.396 acc: 85.91 time: 8.73
[22,   300] loss: 0.404 acc: 85.66 time: 8.65
TESTING:
Accuracy of the network on the 10000 test images: 82.10 %
Average loss on the 10000 test images: 0.546
[23,   100] loss: 0.370 acc: 87.32 time: 8.79
[23,   200] loss: 0.366 acc: 87.18 time: 8.77
[23,   300] loss: 0.382 acc: 86.42 time: 8.64
TESTING:
Accuracy of the network on the 10000 test images: 82.09 %
Average loss on the 10000 test images: 0.544
```

```
[24,   100] loss: 0.355 acc: 87.47 time: 8.75
[24,   200] loss: 0.353 acc: 87.55 time: 8.79
[24,   300] loss: 0.363 acc: 87.09 time: 8.67
TESTING:
Accuracy of the network on the 10000 test images: 82.49 %
Average loss on the 10000 test images: 0.543
[25,   100] loss: 0.339 acc: 88.34 time: 8.66
[25,   200] loss: 0.347 acc: 87.59 time: 8.73
[25,   300] loss: 0.347 acc: 88.04 time: 8.69
TESTING:
Accuracy of the network on the 10000 test images: 82.38 %
Average loss on the 10000 test images: 0.547
[26,   100] loss: 0.327 acc: 88.53 time: 8.96
[26,   200] loss: 0.330 acc: 88.49 time: 8.91
[26,   300] loss: 0.325 acc: 88.57 time: 8.97
TESTING:
Accuracy of the network on the 10000 test images: 82.58 %
Average loss on the 10000 test images: 0.543
[27,   100] loss: 0.311 acc: 89.08 time: 9.00
[27,   200] loss: 0.332 acc: 88.34 time: 8.97
[27,   300] loss: 0.321 acc: 88.72 time: 8.95
TESTING:
Accuracy of the network on the 10000 test images: 82.68 %
Average loss on the 10000 test images: 0.548
[28,   100] loss: 0.313 acc: 88.84 time: 9.02
[28,   200] loss: 0.308 acc: 88.84 time: 8.94
[28,   300] loss: 0.314 acc: 88.84 time: 8.89
TESTING:
Accuracy of the network on the 10000 test images: 82.67 %
Average loss on the 10000 test images: 0.544
[29,   100] loss: 0.302 acc: 89.34 time: 8.96
[29,   200] loss: 0.307 acc: 89.12 time: 8.89
[29,   300] loss: 0.297 acc: 89.49 time: 8.96
TESTING:
Accuracy of the network on the 10000 test images: 82.70 %
Average loss on the 10000 test images: 0.550
[30,   100] loss: 0.305 acc: 89.33 time: 8.93
[30,   200] loss: 0.316 acc: 89.14 time: 8.91
[30,   300] loss: 0.298 acc: 89.56 time: 8.93
TESTING:
Accuracy of the network on the 10000 test images: 82.75 %
Average loss on the 10000 test images: 0.546
Finished Training
```

```
pretrained = {"20":43.32, "100":61.34, "300":66.11, "1000":75.06,
"3000":80.42, "5000":84.32}
non_pretrained = {"20":24.32, "100":45.25, "300":49.94, "1000":66.25,
"3000":79.48, "5000":82.75}
```

```python
import matplotlib.pyplot as plt

x = [20, 100, 300, 1000, 3000, 5000]

plt.plot(x,pretrained.values(), color = "red", label="pre-trained on 
rotation")
plt.plot(x,non_pretrained.values(), color = "black", label="Without 
pretraining")
plt.legend()
plt.grid(True)
plt.xlim(20,5000)
plt.ylabel("accuracy in %")
plt.xlabel("Samples per class")
plt.show()
```



ImageNette :

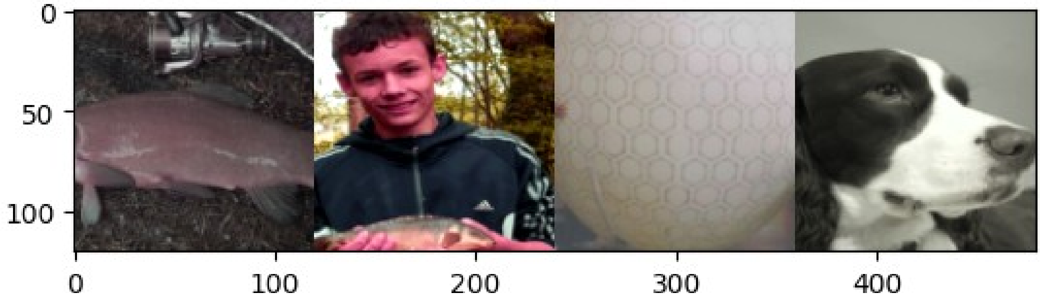
```python
import torch
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import random

import matplotlib.pyplot as plt
import torchvision
import numpy as np


def rotate_img(img, rot):
    if rot == 0:
        return img
    elif rot == 1:
        return transforms.functional.rotate(img, 90)
    elif rot == 2:
        return transforms.functional.rotate(img, 180)
    elif rot == 3:
        return transforms.functional.rotate(img, 270)
    else:
        raise ValueError("Rotation should be 0, 90, 180, or 270
degrees")

class ImagenetteRotation(ImageFolder):
    def __init__(self, root, transform=None):
        super().__init__(root=root, transform=transform)

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

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

        return image, image_rotated,
torch.tensor(rotation_label).long(), torch.tensor(class_label).long()

transform_train = transforms.Compose([
    transforms.RandomResizedCrop(150),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.3, contrast=0.5,
saturation=0.9, hue=0.4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])

transform_test = transforms.Compose([
    transforms.Resize(160),
    transforms.CenterCrop(150),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])

import os
import tarfile
```

```
input_dir = "/kaggle/input/imagenette-full"
dataset_path = os.path.join(input_dir, "imagenette.tgz")
extract_path = "/kaggle/working"

if os.path.exists(dataset_path):
    with tarfile.open(dataset_path, "r:gz") as tar:
        tar.extractall(path=extract_path)
    print(f"Extraction complete! Files are in: {extract_path}")
else:
    print("Dataset not found!")
```

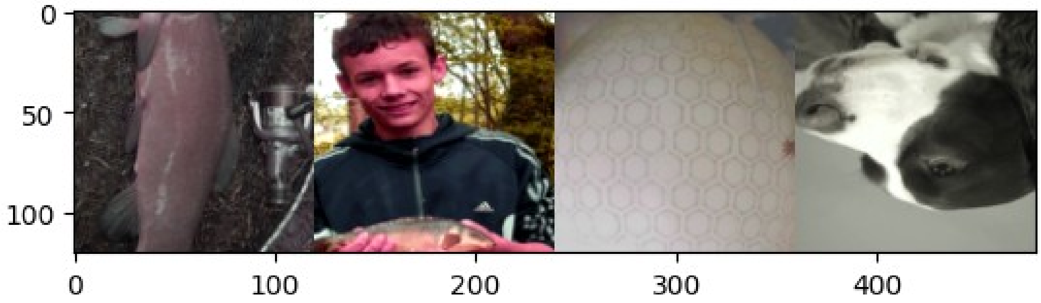
```
Extraction complete! Files are in: /kaggle/working
```

```
batch_size = 64

# trainset =
ImagenetteRotation(root="/kaggle/working/imagenette/train",
transform=transform_train)
trainset = ImagenetteRotation(root="./data/imagenette/train",
transform=transform_train)
trainloader = DataLoader(trainset, batch_size=batch_size,
shuffle=True, num_workers=4, multiprocessing_context='fork')

testset = ImagenetteRotation(root="./data/imagenette/val",
transform=transform_test)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=False,
num_workers=4, multiprocessing_context='fork')

classes = ('Trench', 'English Springer Spaniel', 'Cassette Player',
'Chain Saw', 'Church', 'French Horn', 'Garbage Truck', 'Gas Pump',
'Golf Ball', 'Parachute')
rot_classes = ('0', '90', '180', '270')

# Do Not Run, stores images in system memory and doesn't leave till
the kernel is restarted or all global variables are cleared !!

def imshow(img):
    img = transforms.Normalize((0, 0, 0), (1/0.229, 1/0.224, 1/0.225))
(img)
    img = transforms.Normalize((-0.485, -0.456, -0.406), (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)

img_grid = imshow(torchvision.utils.make_grid(images[:4], padding=0))
```

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



```
Class labels:  Trench Trench Golf Ball English Springer Spaniel
```



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

Rotation Prediction on ImageNette Dataset

```
import time

def run_test(net, testloader, criterion, task):
    correct = 0
    total = 0
    avg_test_loss = 0.0
    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)
```

```
            outputs = net(images)
            predictions = torch.max(outputs,dim=1).indices
            total += labels.shape[0]
            correct += (predictions==labels).sum().item()

            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}')

    return ( round(100*correct/total, 2) > 87.8 )

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

from torchvision.models import efficientnet_b2

net = efficientnet_b2()
net.classifier[1] = nn.Linear(net.classifier[1].in_features, 4)
net = net.to(device)

import torch.optim as optim
from torch.optim.lr_scheduler import CosineAnnealingLR

criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(net.parameters(), lr=0.000005,
weight_decay=1e-8)
scheduler = CosineAnnealingLR(optimizer, T_max=30, eta_min=1e-9)

def train_custom(net, criterion, optimizer, num_epochs, scheduler,
task):

    for epoch in range(num_epochs):

        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):

            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)


            optimizer.zero_grad()

            outputs = net(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            predicted = torch.max(outputs,dim=1).indices

            print_freq = 100
            running_loss += loss.item()

            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()

            if i % print_freq == (print_freq - 1):
                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()

        net.eval()
        if run_test(net,testloader,criterion,task) :
          print("Expected accuracy has been achieved!")
          return

        scheduler.step()

    print('Finished Training')

train_custom(net, criterion, optimizer, num_epochs=80,
scheduler=scheduler, task='rotation')

print('Saving Model ...')
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()},
"./models/imagenette_rotation.pth")
print('Saved Model !')

[1,   100] loss: 1.429 acc: 26.12 time: 48.32
TESTING:
Accuracy of the network on the 10000 test images: 25.00 %
Average loss on the 10000 test images: 1.392
[2,   100] loss: 1.414 acc: 25.00 time: 46.90
TESTING:
Accuracy of the network on the 10000 test images: 25.20 %
```

```
Average loss on the 10000 test images: 1.388
[3,   100] loss: 1.403 acc: 24.95 time: 46.95
TESTING:
Accuracy of the network on the 10000 test images: 26.80 %
Average loss on the 10000 test images: 1.387
[4,   100] loss: 1.400 acc: 24.97 time: 47.37
TESTING:
Accuracy of the network on the 10000 test images: 25.80 %
Average loss on the 10000 test images: 1.395
[5,   100] loss: 1.394 acc: 24.96 time: 46.97
TESTING:
Accuracy of the network on the 10000 test images: 27.20 %
Average loss on the 10000 test images: 1.386
[6,   100] loss: 1.391 acc: 26.26 time: 47.00
TESTING:
Accuracy of the network on the 10000 test images: 31.60 %
Average loss on the 10000 test images: 1.380
[7,   100] loss: 1.378 acc: 31.27 time: 46.27
TESTING:
Accuracy of the network on the 10000 test images: 37.80 %
Average loss on the 10000 test images: 1.323
[8,   100] loss: 1.345 acc: 35.60 time: 47.58
TESTING:
Accuracy of the network on the 10000 test images: 40.60 %
Average loss on the 10000 test images: 1.270
[9,   100] loss: 1.315 acc: 38.60 time: 46.38
TESTING:
Accuracy of the network on the 10000 test images: 43.40 %
Average loss on the 10000 test images: 1.245
[10,   100] loss: 1.300 acc: 39.50 time: 47.54
TESTING:
Accuracy of the network on the 10000 test images: 44.60 %
Average loss on the 10000 test images: 1.243
[11,   100] loss: 1.290 acc: 39.70 time: 46.66
TESTING:
Accuracy of the network on the 10000 test images: 44.00 %
Average loss on the 10000 test images: 1.197
[12,   100] loss: 1.284 acc: 40.20 time: 47.46
TESTING:
Accuracy of the network on the 10000 test images: 46.20 %
Average loss on the 10000 test images: 1.173
[13,   100] loss: 1.274 acc: 40.09 time: 46.50
TESTING:
Accuracy of the network on the 10000 test images: 44.40 %
Average loss on the 10000 test images: 1.187
[14,   100] loss: 1.275 acc: 40.96 time: 46.60
TESTING:
Accuracy of the network on the 10000 test images: 46.00 %
Average loss on the 10000 test images: 1.180
```

```
[15,   100] loss: 1.261 acc: 41.87 time: 45.46
TESTING:
Accuracy of the network on the 10000 test images: 48.80 %
Average loss on the 10000 test images: 1.152
[16,   100] loss: 1.256 acc: 41.98 time: 46.41
TESTING:
Accuracy of the network on the 10000 test images: 49.40 %
Average loss on the 10000 test images: 1.152
[17,   100] loss: 1.245 acc: 42.44 time: 46.03
TESTING:
Accuracy of the network on the 10000 test images: 47.40 %
Average loss on the 10000 test images: 1.157
[18,   100] loss: 1.238 acc: 43.22 time: 47.02
TESTING:
Accuracy of the network on the 10000 test images: 50.20 %
Average loss on the 10000 test images: 1.137
[19,   100] loss: 1.231 acc: 43.42 time: 46.30
TESTING:
Accuracy of the network on the 10000 test images: 49.00 %
Average loss on the 10000 test images: 1.111
[20,   100] loss: 1.230 acc: 43.53 time: 46.97
TESTING:
Accuracy of the network on the 10000 test images: 50.20 %
Average loss on the 10000 test images: 1.115
[21,   100] loss: 1.218 acc: 44.73 time: 46.45
TESTING:
Accuracy of the network on the 10000 test images: 51.00 %
Average loss on the 10000 test images: 1.112
[22,   100] loss: 1.214 acc: 44.68 time: 46.79
TESTING:
Accuracy of the network on the 10000 test images: 50.60 %
Average loss on the 10000 test images: 1.110
[23,   100] loss: 1.211 acc: 45.03 time: 46.41
TESTING:
Accuracy of the network on the 10000 test images: 51.40 %
Average loss on the 10000 test images: 1.084
[24,   100] loss: 1.205 acc: 45.16 time: 46.98
TESTING:
Accuracy of the network on the 10000 test images: 51.00 %
Average loss on the 10000 test images: 1.082
[25,   100] loss: 1.202 acc: 45.55 time: 46.33
TESTING:
Accuracy of the network on the 10000 test images: 54.60 %
Average loss on the 10000 test images: 1.052
[26,   100] loss: 1.187 acc: 46.07 time: 47.29
TESTING:
Accuracy of the network on the 10000 test images: 53.40 %
Average loss on the 10000 test images: 1.064
[27,   100] loss: 1.191 acc: 46.24 time: 46.23
```

```
TESTING:
Accuracy of the network on the 10000 test images: 52.40 %
Average loss on the 10000 test images: 1.059
[28,   100] loss: 1.188 acc: 46.66 time: 47.01
TESTING:
Accuracy of the network on the 10000 test images: 55.00 %
Average loss on the 10000 test images: 1.059
[29,   100] loss: 1.186 acc: 46.26 time: 47.99
TESTING:
Accuracy of the network on the 10000 test images: 53.60 %
Average loss on the 10000 test images: 1.053
[30,   100] loss: 1.187 acc: 45.96 time: 48.15
TESTING:
Accuracy of the network on the 10000 test images: 54.40 %
Average loss on the 10000 test images: 1.045
[31,   100] loss: 1.185 acc: 46.26 time: 47.47
TESTING:
Accuracy of the network on the 10000 test images: 55.00 %
Average loss on the 10000 test images: 1.044
[32,   100] loss: 1.189 acc: 46.17 time: 46.93
TESTING:
Accuracy of the network on the 10000 test images: 52.80 %
Average loss on the 10000 test images: 1.055
[33,   100] loss: 1.185 acc: 46.74 time: 46.93
TESTING:
Accuracy of the network on the 10000 test images: 53.80 %
Average loss on the 10000 test images: 1.050
[34,   100] loss: 1.192 acc: 46.34 time: 48.22
TESTING:
Accuracy of the network on the 10000 test images: 52.00 %
Average loss on the 10000 test images: 1.057
[35,   100] loss: 1.182 acc: 46.62 time: 48.43
TESTING:
Accuracy of the network on the 10000 test images: 53.20 %
Average loss on the 10000 test images: 1.053
[36,   100] loss: 1.182 acc: 46.41 time: 47.50
TESTING:
Accuracy of the network on the 10000 test images: 54.00 %
Average loss on the 10000 test images: 1.030
[37,   100] loss: 1.186 acc: 46.05 time: 48.13
TESTING:
Accuracy of the network on the 10000 test images: 53.40 %
Average loss on the 10000 test images: 1.053
[38,   100] loss: 1.187 acc: 46.71 time: 46.99
TESTING:
Accuracy of the network on the 10000 test images: 54.00 %
Average loss on the 10000 test images: 1.037
[39,   100] loss: 1.181 acc: 46.49 time: 47.10
TESTING:
```

```
Accuracy of the network on the 10000 test images: 52.80 %
Average loss on the 10000 test images: 1.055
[40,   100] loss: 1.180 acc: 47.27 time: 46.50
TESTING:
Accuracy of the network on the 10000 test images: 54.20 %
Average loss on the 10000 test images: 1.058
[41,   100] loss: 1.178 acc: 46.95 time: 47.54
TESTING:
Accuracy of the network on the 10000 test images: 53.80 %
Average loss on the 10000 test images: 1.062
[42,   100] loss: 1.183 acc: 46.66 time: 47.21
TESTING:
Accuracy of the network on the 10000 test images: 54.00 %
Average loss on the 10000 test images: 1.055
[43,   100] loss: 1.181 acc: 47.34 time: 48.39
TESTING:
Accuracy of the network on the 10000 test images: 53.60 %
Average loss on the 10000 test images: 1.068
[44,   100] loss: 1.184 acc: 47.03 time: 46.61
TESTING:
Accuracy of the network on the 10000 test images: 54.80 %
Average loss on the 10000 test images: 1.054
[45,   100] loss: 1.182 acc: 47.07 time: 46.18
TESTING:
Accuracy of the network on the 10000 test images: 57.80 %
Average loss on the 10000 test images: 1.022
[46,   100] loss: 1.174 acc: 47.59 time: 46.00
TESTING:
Accuracy of the network on the 10000 test images: 52.00 %
Average loss on the 10000 test images: 1.083
[47,   100] loss: 1.172 acc: 48.05 time: 46.60
TESTING:
Accuracy of the network on the 10000 test images: 55.80 %
Average loss on the 10000 test images: 1.058
[48,   100] loss: 1.172 acc: 47.89 time: 46.59
TESTING:
Accuracy of the network on the 10000 test images: 56.40 %
Average loss on the 10000 test images: 1.035
[49,   100] loss: 1.167 acc: 47.92 time: 47.28
TESTING:
Accuracy of the network on the 10000 test images: 54.20 %
Average loss on the 10000 test images: 1.022
[50,   100] loss: 1.160 acc: 49.19 time: 46.36
TESTING:
Accuracy of the network on the 10000 test images: 57.00 %
Average loss on the 10000 test images: 1.012
[51,   100] loss: 1.159 acc: 48.86 time: 47.05
TESTING:
Accuracy of the network on the 10000 test images: 59.20 %
```

```
Average loss on the 10000 test images: 0.993
[52,   100] loss: 1.151 acc: 48.82 time: 47.43
TESTING:
Accuracy of the network on the 10000 test images: 59.40 %
Average loss on the 10000 test images: 1.003
[53,   100] loss: 1.146 acc: 49.46 time: 46.10
TESTING:
Accuracy of the network on the 10000 test images: 60.20 %
Average loss on the 10000 test images: 0.976
[54,   100] loss: 1.134 acc: 50.27 time: 46.65
TESTING:
Accuracy of the network on the 10000 test images: 58.60 %
Average loss on the 10000 test images: 0.968
[55,   100] loss: 1.128 acc: 50.80 time: 47.42
TESTING:
Accuracy of the network on the 10000 test images: 62.40 %
Average loss on the 10000 test images: 0.903
[56,   100] loss: 1.123 acc: 51.16 time: 47.05
TESTING:
Accuracy of the network on the 10000 test images: 58.00 %
Average loss on the 10000 test images: 0.978
[57,   100] loss: 1.106 acc: 51.84 time: 46.67
TESTING:
Accuracy of the network on the 10000 test images: 62.20 %
Average loss on the 10000 test images: 0.914
[58,   100] loss: 1.095 acc: 52.60 time: 46.76
TESTING:
Accuracy of the network on the 10000 test images: 62.60 %
Average loss on the 10000 test images: 0.921
[59,   100] loss: 1.088 acc: 52.80 time: 46.18
TESTING:
Accuracy of the network on the 10000 test images: 64.40 %
Average loss on the 10000 test images: 0.896
[60,   100] loss: 1.077 acc: 53.56 time: 46.99
TESTING:
Accuracy of the network on the 10000 test images: 64.40 %
Average loss on the 10000 test images: 0.908
[61,   100] loss: 1.073 acc: 53.68 time: 45.40
TESTING:
Accuracy of the network on the 10000 test images: 66.00 %
Average loss on the 10000 test images: 0.901
[62,   100] loss: 1.064 acc: 53.81 time: 46.64
TESTING:
Accuracy of the network on the 10000 test images: 69.00 %
Average loss on the 10000 test images: 0.853
[63,   100] loss: 1.042 acc: 55.66 time: 45.63
TESTING:
Accuracy of the network on the 10000 test images: 67.80 %
Average loss on the 10000 test images: 0.847
```

```
[64,   100] loss: 1.035 acc: 55.95 time: 46.23
TESTING:
Accuracy of the network on the 10000 test images: 67.20 %
Average loss on the 10000 test images: 0.854
[65,   100] loss: 1.022 acc: 56.12 time: 46.18
TESTING:
Accuracy of the network on the 10000 test images: 69.60 %
Average loss on the 10000 test images: 0.845
[66,   100] loss: 1.022 acc: 56.51 time: 47.01
TESTING:
Accuracy of the network on the 10000 test images: 68.00 %
Average loss on the 10000 test images: 0.822
[67,   100] loss: 0.991 acc: 58.16 time: 46.79
TESTING:
Accuracy of the network on the 10000 test images: 70.20 %
Average loss on the 10000 test images: 0.798
[68,   100] loss: 0.987 acc: 58.23 time: 46.94
TESTING:
Accuracy of the network on the 10000 test images: 71.00 %
Average loss on the 10000 test images: 0.782
[69,   100] loss: 0.980 acc: 58.53 time: 46.64
TESTING:
Accuracy of the network on the 10000 test images: 72.60 %
Average loss on the 10000 test images: 0.763
[70,   100] loss: 0.968 acc: 59.56 time: 46.42
TESTING:
Accuracy of the network on the 10000 test images: 71.60 %
Average loss on the 10000 test images: 0.739
[71,   100] loss: 0.955 acc: 60.09 time: 46.25
TESTING:
Accuracy of the network on the 10000 test images: 70.00 %
Average loss on the 10000 test images: 0.767
[72,   100] loss: 0.942 acc: 61.00 time: 46.38
TESTING:
Accuracy of the network on the 10000 test images: 73.40 %
Average loss on the 10000 test images: 0.714
[73,   100] loss: 0.924 acc: 61.80 time: 46.26
TESTING:
Accuracy of the network on the 10000 test images: 70.40 %
Average loss on the 10000 test images: 0.732
[74,   100] loss: 0.911 acc: 61.80 time: 46.97
TESTING:
Accuracy of the network on the 10000 test images: 73.00 %
Average loss on the 10000 test images: 0.689
[75,   100] loss: 0.903 acc: 61.98 time: 45.72
TESTING:
Accuracy of the network on the 10000 test images: 75.00 %
Average loss on the 10000 test images: 0.679
[76,   100] loss: 0.898 acc: 62.59 time: 47.19
```

```
TESTING:
Accuracy of the network on the 10000 test images: 73.40 %
Average loss on the 10000 test images: 0.665
[77,   100] loss: 0.880 acc: 63.42 time: 46.00
TESTING:
Accuracy of the network on the 10000 test images: 75.40 %
Average loss on the 10000 test images: 0.658
[78,   100] loss: 0.863 acc: 64.55 time: 47.22
TESTING:
Accuracy of the network on the 10000 test images: 76.00 %
Average loss on the 10000 test images: 0.641
[79,   100] loss: 0.865 acc: 64.14 time: 45.62
TESTING:
Accuracy of the network on the 10000 test images: 76.60 %
Average loss on the 10000 test images: 0.609
[80,   100] loss: 0.852 acc: 64.96 time: 46.97
TESTING:
Accuracy of the network on the 10000 test images: 77.40 %
Average loss on the 10000 test images: 0.597
Finished Training
Saving Model ...

----------------------------------------------------------------------
-----
RuntimeError                              Traceback (most recent call
last)
<ipython-input-11-f9941c17cd48> in <cell line: 5>()
      3 print('Saving Model ...')
      4 # TODO: Save the model
----> 5 torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()},
"./models/imagenette_rotation.pth")
      6 print('Saved Model !')

/usr/local/lib/python3.10/dist-packages/torch/serialization.py in
save(obj, f, pickle_module, pickle_protocol,
_use_new_zipfile_serialization, _disable_byteorder_record)
    847
    848     if _use_new_zipfile_serialization:
--> 849         with _open_zipfile_writer(f) as opened_zipfile:
    850             _save(
    851                 obj,

/usr/local/lib/python3.10/dist-packages/torch/serialization.py in
_open_zipfile_writer(name_or_buffer)
    714     else:
    715         container = _open_zipfile_writer_buffer
--> 716     return container(name_or_buffer)
    717
    718
```

```
/usr/local/lib/python3.10/dist-packages/torch/serialization.py in
__init__(self, name)
    685
super().__init__(torch._C.PyTorchFileWriter(self.file_stream))
    686         else:
--> 687
super().__init__(torch._C.PyTorchFileWriter(self.name))
    688
    689     def __exit__(self, *args) -> None:

RuntimeError: Parent directory ./models does not exist.
```

```
train_custom(net, criterion, optimizer, num_epochs=30,
scheduler=scheduler, task='rotation')
```

```
[1,   100] loss: 0.535 acc: 78.58 time: 27.79
[1,   200] loss: 0.551 acc: 77.88 time: 27.54
TESTING:
Accuracy of the network on the 10000 test images: 87.20 %
Average loss on the 10000 test images: 0.344
[2,   100] loss: 0.542 acc: 78.69 time: 27.80
[2,   200] loss: 0.530 acc: 78.97 time: 27.55
TESTING:
Accuracy of the network on the 10000 test images: 87.20 %
Average loss on the 10000 test images: 0.355
[3,   100] loss: 0.542 acc: 78.48 time: 27.86
[3,   200] loss: 0.551 acc: 78.09 time: 27.67
TESTING:
Accuracy of the network on the 10000 test images: 86.80 %
Average loss on the 10000 test images: 0.366
[4,   100] loss: 0.552 acc: 77.88 time: 27.84
[4,   200] loss: 0.532 acc: 78.94 time: 27.56
TESTING:
Accuracy of the network on the 10000 test images: 87.40 %
Average loss on the 10000 test images: 0.349
[5,   100] loss: 0.531 acc: 79.12 time: 27.76
[5,   200] loss: 0.568 acc: 77.91 time: 27.57
TESTING:
Accuracy of the network on the 10000 test images: 88.60 %
Average loss on the 10000 test images: 0.347
Expected accuracy has been achieved!
```

```
print('Saving Model ...')
torch.save({"parameters":net.state_dict(),
"optimizer":optimizer.state_dict()},
"./models/imagenette_rotation.pth")
print('Saved Model !')
```

```
Saving Model ...
Saved Model !
```

```
checkpoint =
torch.load("./models/imagenette_rotation.pth",map_location=torch.devic
e('mps'))
net.load_state_dict(checkpoint['parameters'])
# optimizer.load_state_dict(checkpoint['optimizer'])

net.eval()
run_test(net,testloader,criterion,"rotation")
```

```
TESTING:
Accuracy of the network on the 10000 test images: 87.40 %
Average loss on the 10000 test images: 0.346
```

```
False
```