In [2]:
```
'''ResNet in PyTorch.

For Pre-activation ResNet, see 'preact_resnet.py'.

Reference:
[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
    Deep Residual Learning for Image Recognition. arXiv:1512.03385
'''
import torch
import torch.nn as nn
import torch.nn.functional as F
```

In [3]:
```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1, prob_dropout=0.3):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias
        self.bn1 = nn.BatchNorm2d(planes)

        # mod here
        self.dropout1 = nn.Dropout2d(p=prob_dropout)

        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)


        # mod here
        self.dropout2 = nn.Dropout2d(p=prob_dropout)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):

        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

In [4]:
```
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
```

```
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion *
                               planes, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion*planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

In [5]:

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
```

```
        out = self.linear(out)
        return out
```

In [6]:
```
def ResNet9():
    return ResNet(BasicBlock, [1, 1, 1, 1])

def ResNet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])


# def ResNet34():
#     return ResNet(BasicBlock, [3, 4, 6, 3])


# def ResNet50():
#     return ResNet(Bottleneck, [3, 4, 6, 3])


# def ResNet101():
#     return ResNet(Bottleneck, [3, 4, 23, 3])


# def ResNet152():
#     return ResNet(Bottleneck, [3, 8, 36, 3])


# def test():
#     net = ResNet18()
#     y = net(torch.randn(1, 3, 32, 32))
#     print(y.size())

# test()

print("done")
```

```
done
```

In [7]:
```
from torchvision import transforms, datasets
import torch.utils.data
from torchvision.transforms import ToTensor
```

In [8]:
```
torch.manual_seed(1024)
# Define transformations to be applied on the training dataset
train_transform = transforms.Compose([
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue
    transforms.RandomHorizontalFlip(), #apply horizontal flipping
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(), # Randomly flip the images on the hor
    transforms.RandomRotation(10), # Randomly rotate the images by +/- 10 de
    transforms.RandomCrop(32, padding=4), # Apply random crops
    transforms.ToTensor(), # Convert images to PyTorch tensors
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), # Normalize imag
])

# Define transformations to be applied on the testing dataset
test_transform = transforms.Compose([
```

```
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

# Import the datasets and transform it
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, t
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, t

# Define the batch size for training
batch_size = 16

# Create DataLoader
trainDataLoader = torch.utils.data.DataLoader(train_dataset, batch_size=batc
testDataLoader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_

# for batch in train_loader:
#     print(batch)
print("done")
```

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

In [9]:
```
# Import all the necessary libraries

import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pickle
from PIL import Image
import matplotlib.pyplot as plt

# Set a manual seed for reproducibility
torch.manual_seed(1024)
```

Out[9]:
```
<torch._C.Generator at 0x14f8511f2cd0>
```

In [10]:
```
# Function used to calculate the accuracy on the cifar_no_labels dataset
def calculate_accuracy(predictions):
    actual = []
    pattern = [8, 2, 9, 0, 4, 3, 6, 1, 7, 5]
    for num in pattern:
        actual.extend([num] * 1000)
    num_matches = sum(1 for pred, act in zip(predictions, actual) if pred ==
    accuracy = (num_matches / len(predictions)) * 100
    return accuracy
```

In [11]:
```
# Function to load and transform the cifar_no_labels dataset
def load_kaggle_data(file, transform=None):
    import pickle
    with open(file, 'rb') as fo:
        batch = pickle.load(fo, encoding='bytes')
    images = batch[b'data']
```

```
    images = images.reshape((-1, 3, 32, 32)).transpose(0, 2, 3, 1)
    min_val = np.min(images)
    max_val = np.max(images)
    images = (images - min_val) / (max_val - min_val) #normalizing facepalm
    processed_images=[]
    for i,image in enumerate(images):
        pil_image = Image.fromarray((image * 255).astype(np.uint8))
        # Apply transformation
        pil_image = transform(pil_image)

        # Append transformed image
        processed_images.append(pil_image)
    return processed_images
```

In [12]:
```
# Tranformation to be applied on the cifar_test_nolabels.pkl dataset
kaggle_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Loading the cifar_test_nolabels images and transforming as needed
test_images = load_kaggle_data('kaggle/cifar_test_nolabels.pkl', transform=k
kaggleLoader = torch.utils.data.DataLoader(test_images, batch_size=1,shuffle
print(len(kaggleLoader))
```

```
10000
```

In [13]:
```
import torch.optim.lr_scheduler as lr_scheduler

# Initialize empty lists to store training and test loss history
train_loss_history = []
test_loss_history = []
train_accuracy_history = []
test_accuracy_history = []

no_label_accuracy_history = []

# Initialize variables to track accuracy
matches = 0
total = 0

# Initialize the best Kaggle accuracy and learning rate
best_kagg_acc = 0.0
lr_best = -1

# Define a list of learning rates to try (commented out for now)
# lr_arr = [.003, .005, 0.0001, 0.0005, 0.0002, 0.0003, 0.0004, 0.005, 0.008

# Iterate through the learning rates (commented out for now)
# for learn_rate in lr_arr[:2]:

# Set the learning rate (using a fixed value for now)
learn_rate = .004
```

In [14]:
```
# Set the number of training epochs
```

```
num_epochs = 60
```

In [15]:
```
# Initialize model, loss function, optimizer, and learning rate scheduler
model = ResNet9().cuda() # .cuda to train on the GPU
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learn_rate, momentum=0.9,
scheduler = lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs)  # T
```

In [16]:
```
# Print a message to indicate the start of the training loop
print("Beginning training arc now!~~~ \n")

for epoch in range(num_epochs):
    train_loss = 0.0
    test_loss = 0.0
    train_acc = 0.0
    test_acc = 0.0

    model.train()
    for i, data in enumerate(trainDataLoader):
        images, labels = data
        images = images.cuda()
        labels = labels.cuda()
        optimizer.zero_grad() # zero out any gradient values from the previo
        predicted_output = model(images) # forward propagation
        fit = loss(predicted_output, labels)  # calculate our measure of goo
        fit.backward() # backpropagation
        optimizer.step() # update the weights of our trainable parameters
        train_loss += fit.item()
        _, predicted = torch.max(predicted_output.data, 1)
        train_acc += (predicted == labels).sum().item()

    scheduler.step()
    for i, data in  enumerate(testDataLoader):
        with torch.no_grad():
            images, labels = data
            images = images.cuda()
            labels = labels.cuda()
            predicted_output = model(images)
            fit = loss(predicted_output, labels)
            test_loss += fit.item()
            _, predicted = torch.max(predicted_output.data, 1)
            matches += (predicted == labels).sum().item()
            total += labels.size(0)
            test_acc += (predicted == labels).sum().item()
    train_loss = train_loss / len(trainDataLoader)
    test_loss = test_loss / len(testDataLoader)
    train_acc = train_acc / len(trainDataLoader.dataset)
    test_acc = test_acc / len(testDataLoader.dataset)
    train_loss_history += [train_loss]
    test_loss_history += [test_loss]
    train_accuracy_history += [train_acc]
    test_accuracy_history += [test_acc]

    predictions = []
    # model.eval()
```

```
    # Saving the best model if it exceeds the previous best accuracy
    for image in kaggleLoader:
        image = image.cuda()
        with torch.no_grad():
            output = model(image)
        predictions.append(output.argmax().item())
    curr_acc = calculate_accuracy(predictions)
    if curr_acc > best_kagg_acc:
        best_kagg_acc = curr_acc
        torch.save(model.state_dict(), 'best_resnet9_model.pth')
    no_label_accuracy_history += [curr_acc]

    print(f'Epoch {epoch}, Train loss {train_loss}, Train accuracy {train_ac
```

```
Beginning training arc now!~~~

Epoch 0, Train loss 1.565065152053833, Train accuracy 0.429, Test loss 1.409
7601428985596, Test accuracy 0.502, NoLabel Accuracy 17.11
Epoch 1, Train loss 1.1578211816215516, Train accuracy 0.58694, Test loss 1.
187808895111084, Test accuracy 0.5829, NoLabel Accuracy 28.83
Epoch 2, Train loss 0.9826439865589142, Train accuracy 0.6525, Test loss 1.0
514758522987366, Test accuracy 0.6435, NoLabel Accuracy 22.509999999999998
Epoch 3, Train loss 0.8553131859731674, Train accuracy 0.70014, Test loss 0.
9139261562108993, Test accuracy 0.6875, NoLabel Accuracy 35.06
Epoch 4, Train loss 0.7726250850772858, Train accuracy 0.72884, Test loss 0.
8589714540958404, Test accuracy 0.7147, NoLabel Accuracy 36.25
Epoch 5, Train loss 0.7130328175830841, Train accuracy 0.75164, Test loss 0.
7514945923805236, Test accuracy 0.7394, NoLabel Accuracy 38.76
Epoch 6, Train loss 0.6559436442351341, Train accuracy 0.77198, Test loss 0.
7745088212013245, Test accuracy 0.7364, NoLabel Accuracy 37.980000000000004
Epoch 7, Train loss 0.6150526032543182, Train accuracy 0.7879, Test loss 0.7
079473152637482, Test accuracy 0.7578, NoLabel Accuracy 40.71
Epoch 8, Train loss 0.5791047469520569, Train accuracy 0.79818, Test loss 0.
6445163821697235, Test accuracy 0.7767, NoLabel Accuracy 45.45
Epoch 9, Train loss 0.5449702680659294, Train accuracy 0.81236, Test loss 0.
6579202128052711, Test accuracy 0.7776, NoLabel Accuracy 40.75
Epoch 10, Train loss 0.5274680589437485, Train accuracy 0.8159, Test loss 0.
5718300176024437, Test accuracy 0.8066, NoLabel Accuracy 45.69
Epoch 11, Train loss 0.4985662501525879, Train accuracy 0.82592, Test loss
0.5657376887559891, Test accuracy 0.8079, NoLabel Accuracy 48.870000000000005
Epoch 12, Train loss 0.47443469311118125, Train accuracy 0.83582, Test loss
0.5632015392065048, Test accuracy 0.8084, NoLabel Accuracy 46.6
Epoch 13, Train loss 0.45703014152288435, Train accuracy 0.84052, Test loss
0.5654279923379422, Test accuracy 0.8092, NoLabel Accuracy 48.85
Epoch 14, Train loss 0.4362173969078064, Train accuracy 0.8491, Test loss 0.
5220353974461556, Test accuracy 0.8257, NoLabel Accuracy 50.44
Epoch 15, Train loss 0.4160895113795996, Train accuracy 0.8546, Test loss 0.
5670471617817878, Test accuracy 0.8082, NoLabel Accuracy 49.09
Epoch 16, Train loss 0.40798100676357746, Train accuracy 0.8587, Test loss
0.5184087174713612, Test accuracy 0.8282, NoLabel Accuracy 50.32
Epoch 17, Train loss 0.38880334553897383, Train accuracy 0.86464, Test loss
0.5005707773327828, Test accuracy 0.8287, NoLabel Accuracy 47.21
Epoch 18, Train loss 0.3745120352858305, Train accuracy 0.87084, Test loss
0.4803273251593113, Test accuracy 0.8398, NoLabel Accuracy 48.94
Epoch 19, Train loss 0.36028610295176505, Train accuracy 0.87342, Test loss
0.507972255641222, Test accuracy 0.8315, NoLabel Accuracy 49.76
Epoch 20, Train loss 0.3497142912572622, Train accuracy 0.87758, Test loss
0.4499871678709984, Test accuracy 0.8517, NoLabel Accuracy 53.68000000000001
Epoch 21, Train loss 0.3335770726078749, Train accuracy 0.88486, Test loss
0.45198090255856516, Test accuracy 0.8501, NoLabel Accuracy 53.21
Epoch 22, Train loss 0.322429395595789, Train accuracy 0.88634, Test loss 0.
45899347985386846, Test accuracy 0.8446, NoLabel Accuracy 54.0
Epoch 23, Train loss 0.31269431653410196, Train accuracy 0.89018, Test loss
0.4456697785943747, Test accuracy 0.8498, NoLabel Accuracy 52.64
Epoch 24, Train loss 0.3030749848395586, Train accuracy 0.89318, Test loss
0.4614305145442486, Test accuracy 0.8495, NoLabel Accuracy 52.96999999999999
Epoch 25, Train loss 0.29022502734482286, Train accuracy 0.8975, Test loss
0.4573403076648712, Test accuracy 0.8487, NoLabel Accuracy 52.459999999999994
```

```
Epoch 26, Train loss 0.283657560082674, Train accuracy 0.901, Test loss 0.42
907574498951434, Test accuracy 0.8603, NoLabel Accuracy 55.44
Epoch 27, Train loss 0.2726796293652058, Train accuracy 0.90508, Test loss
0.4192110553309321, Test accuracy 0.8619, NoLabel Accuracy 55.76999999999999
6
Epoch 28, Train loss 0.2641718540802598, Train accuracy 0.90782, Test loss
0.44235115008056164, Test accuracy 0.8552, NoLabel Accuracy 53.3399999999999
96
Epoch 29, Train loss 0.24958603388398887, Train accuracy 0.91146, Test loss
0.41934661067724227, Test accuracy 0.8616, NoLabel Accuracy 51.4599999999999
94
Epoch 30, Train loss 0.2469574574407935, Train accuracy 0.91274, Test loss
0.43635995832383634, Test accuracy 0.8558, NoLabel Accuracy 52.33
Epoch 31, Train loss 0.23779281569600105, Train accuracy 0.91778, Test loss
0.43050874363482, Test accuracy 0.8557, NoLabel Accuracy 53.6
Epoch 32, Train loss 0.2292194628391415, Train accuracy 0.9195, Test loss 0.
4057857525393367, Test accuracy 0.8646, NoLabel Accuracy 54.35
Epoch 33, Train loss 0.22133019650708885, Train accuracy 0.92218, Test loss
0.39317816500514746, Test accuracy 0.8694, NoLabel Accuracy 56.8200000000000
1
Epoch 34, Train loss 0.21131700300335884, Train accuracy 0.925, Test loss 0.
4045492551088333, Test accuracy 0.8713, NoLabel Accuracy 56.00000000000001
Epoch 35, Train loss 0.20280368482761085, Train accuracy 0.92898, Test loss
0.4001813688233495, Test accuracy 0.8659, NoLabel Accuracy 56.66
Epoch 36, Train loss 0.19795199086487295, Train accuracy 0.92934, Test loss
0.3875773511737585, Test accuracy 0.8736, NoLabel Accuracy 56.7200000000000
6
Epoch 37, Train loss 0.19136814153894782, Train accuracy 0.93328, Test loss
0.40164287349283695, Test accuracy 0.8699, NoLabel Accuracy 56.8899999999999
9
Epoch 38, Train loss 0.18078936842478813, Train accuracy 0.93656, Test loss
0.3980545503363013, Test accuracy 0.8709, NoLabel Accuracy 55.34
Epoch 39, Train loss 0.1762613302589953, Train accuracy 0.93902, Test loss
0.38579977177381514, Test accuracy 0.8755, NoLabel Accuracy 55.7799999999999
94
Epoch 40, Train loss 0.16939530287876725, Train accuracy 0.94044, Test loss
0.397768667177856, Test accuracy 0.8697, NoLabel Accuracy 55.269999999999996
Epoch 41, Train loss 0.16668343320444226, Train accuracy 0.94134, Test loss
0.38964408542886375, Test accuracy 0.8741, NoLabel Accuracy 55.75
Epoch 42, Train loss 0.16011176820661874, Train accuracy 0.94454, Test loss
0.38756698313578963, Test accuracy 0.8736, NoLabel Accuracy 54.36
Epoch 43, Train loss 0.15396924899458886, Train accuracy 0.94588, Test loss
0.38111162870526316, Test accuracy 0.8756, NoLabel Accuracy 56.54
Epoch 44, Train loss 0.14712750981446357, Train accuracy 0.94794, Test loss
0.3846097068145871, Test accuracy 0.8777, NoLabel Accuracy 55.94
Epoch 45, Train loss 0.14395980868637562, Train accuracy 0.94946, Test loss
0.37511624975204466, Test accuracy 0.8776, NoLabel Accuracy 55.19
Epoch 46, Train loss 0.1410141050895676, Train accuracy 0.95124, Test loss
0.36263283863365653, Test accuracy 0.8823, NoLabel Accuracy 56.4899999999999
95
Epoch 47, Train loss 0.13488670786011964, Train accuracy 0.95336, Test loss
0.37289379360228775, Test accuracy 0.8795, NoLabel Accuracy 55.87
Epoch 48, Train loss 0.13136343899350614, Train accuracy 0.954, Test loss 0.
3748378746725619, Test accuracy 0.8789, NoLabel Accuracy 55.669999999999995
Epoch 49, Train loss 0.1308792077526264, Train accuracy 0.95426, Test loss
0.3734875489644706, Test accuracy 0.8799, NoLabel Accuracy 55.41000000000000
```

```
4
Epoch 50, Train loss 0.12566706750977785, Train accuracy 0.95676, Test loss
0.37194543466791513, Test accuracy 0.8821, NoLabel Accuracy 56.8999999999999
Epoch 51, Train loss 0.12422420908432454, Train accuracy 0.95728, Test loss
0.37819664292261007, Test accuracy 0.8776, NoLabel Accuracy 56.43
Epoch 52, Train loss 0.12539945043623446, Train accuracy 0.9577, Test loss
0.37442457415908575, Test accuracy 0.8824, NoLabel Accuracy 56.42
Epoch 53, Train loss 0.12033280499543994, Train accuracy 0.95804, Test loss
0.3682648988887668, Test accuracy 0.8821, NoLabel Accuracy 56.42
Epoch 54, Train loss 0.12072094284471124, Train accuracy 0.95826, Test loss
0.3634823924150318, Test accuracy 0.884, NoLabel Accuracy 56.18
Epoch 55, Train loss 0.11805840330667794, Train accuracy 0.95942, Test loss
0.37241467096805575, Test accuracy 0.8809, NoLabel Accuracy 56.46
Epoch 56, Train loss 0.11670963669043034, Train accuracy 0.95898, Test loss
0.3693078789971769, Test accuracy 0.8838, NoLabel Accuracy 56.57
Epoch 57, Train loss 0.11667076341520995, Train accuracy 0.96014, Test loss
0.35885330810397864, Test accuracy 0.8843, NoLabel Accuracy 56.7200000000000
06
Epoch 58, Train loss 0.11490545755714178, Train accuracy 0.96076, Test loss
0.3722071375776082, Test accuracy 0.8824, NoLabel Accuracy 56.64
Epoch 59, Train loss 0.11323038229238241, Train accuracy 0.96176, Test loss
0.36542561520338057, Test accuracy 0.8844, NoLabel Accuracy 56.65
```

In [29]:
```
# No time to implement the model checkppoint code as it is not asked in the

# Define the checkpoint directory
# checkpoint_dir = "checkpoints"
# os.makedirs(checkpoint_dir, exist_ok=True)

# # Training loop
# for epoch in range(num_epochs):
#     # Training code here...

#     # Save a checkpoint
#     checkpoint = {
#         "epoch": epoch,
#         "model_state_dict": model.state_dict(),
#         "optimizer_state_dict": optimizer.state_dict(),
#         "loss": loss,
#     }
#     torch.save(checkpoint, os.path.join(checkpoint_dir, f"checkpoint_{epoc
```

In [17]:
```
model.eval()
```

Out[17]:
```
ResNet(
  (conv1): Conv2d(3, 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_runnin
g_stats=True)
  (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_ru
nning_stats=True)
      (dropout1): Dropout2d(p=0.3, inplace=False)
      (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_ru
nning_stats=True)
      (dropout2): Dropout2d(p=0.3, inplace=False)
      (shortcut): Sequential()
    )
  )
  (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_r
unning_stats=True)
      (dropout1): Dropout2d(p=0.3, inplace=False)
      (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_r
unning_stats=True)
      (dropout2): Dropout2d(p=0.3, inplace=False)
      (shortcut): 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_r
unning_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_r
unning_stats=True)
      (dropout1): Dropout2d(p=0.3, inplace=False)
      (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_r
unning_stats=True)
      (dropout2): Dropout2d(p=0.3, inplace=False)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
```

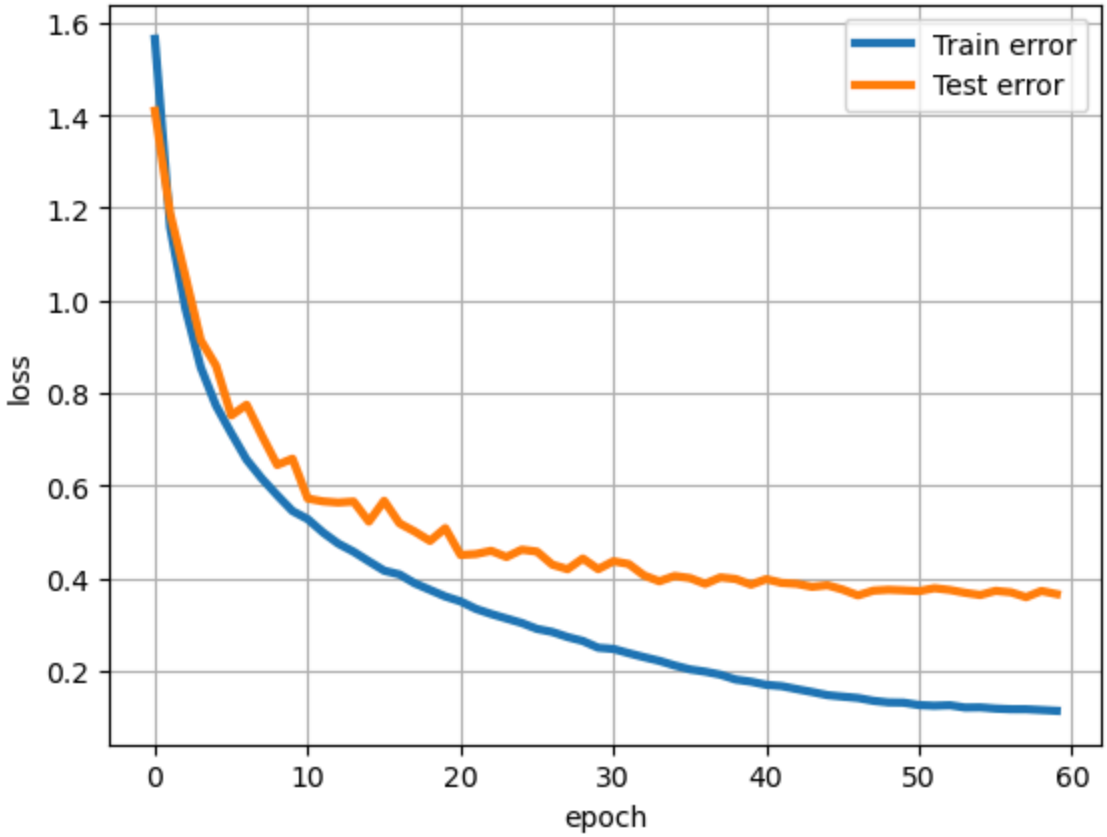
```
      )
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (dropout1): Dropout2d(p=0.3, inplace=False)
      (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_r
unning_stats=True)
      (dropout2): Dropout2d(p=0.3, inplace=False)
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
    )
  )
  (linear): Linear(in_features=512, out_features=10, bias=True)
)
```

In [18]:
```
predictions = []
# model.eval()
for image in kaggleLoader:
    image = image.cuda()
    with torch.no_grad():
        output = model(image)
    predictions.append(output.argmax().item())
curr_acc = calculate_accuracy(predictions)
```

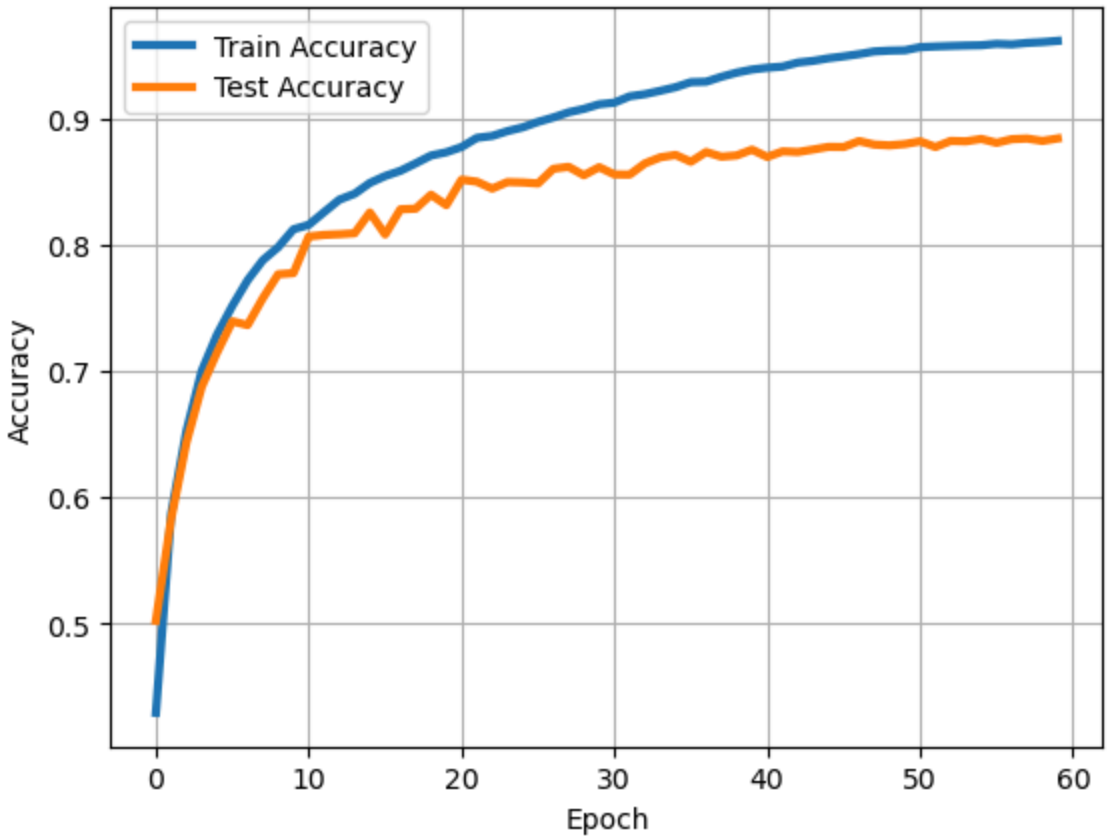
In [19]:
```
torch.save(model.state_dict(), 'learn_rate_consistent_model.pth')
name_file = "learn_rate_const_lr_" + str(learn_rate) + ".txt"
with open(name_file, 'w') as file:
    for row in [train_loss_history, test_loss_history]:
        file.write('\t'.join(map(str, row)) + '\n')
print("finished round " + str(learn_rate) + " with ACC of " + str(curr_acc))
print("done")
```

```
finished round 0.004 with ACC of 66.22
done
```

In [20]:
```
plt.plot(range(len(train_loss_history)),train_loss_history,'-',linewidth=3,l
plt.plot(range(len(test_loss_history)),test_loss_history,'-',linewidth=3,lab
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
plt.show()
```



In [21]:
```
plt.plot(range(len(train_accuracy_history)),train_accuracy_history,'-',linew
plt.plot(range(len(test_accuracy_history)),test_accuracy_history,'-',linewic
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()
plt.show()
```



In [32]:
```
model.load_state_dict(torch.load('learn_rate_consistent_model.pth'))
```

Out[32]: <All keys matched successfully>

In [33]:
```
correct = 0
total = 0
model.eval()
for data in testDataLoader:
    images, labels = data
    images = images.cuda()
    labels = labels.cuda()
    with torch.no_grad():
        predicted_output = model(images)
        _, predicted = torch.max(predicted_output.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print('Accuracy of the network on the test images: %d %%' % (100 * accuracy))
```

```
Accuracy of the network on the test images: 92 %
```

In [22]:
```
def count_parameters(model):
    """
    Counts the number of trainable and non-trainable parameters in a PyTorch
    """
    total_params = sum(p.numel() for p in model.parameters())
    trainable_params = sum(p.numel() for p in model.parameters() if p.requir
    non_trainable_params = total_params - trainable_params
```

```
    print(f"Total parameters: {total_params}")
    print(f"Trainable parameters: {trainable_params}")
    print(f"Non-trainable parameters: {non_trainable_params}")
```

In [28]:
```
count_parameters(model)
```

```
Total parameters: 4903242
Trainable parameters: 4903242
Non-trainable parameters: 0
```

In [34]:
```
pip install nbconvert
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Requirement already satisfied: nbconvert in /share/apps/anaconda3/2024.02/li
b/python3.11/site-packages (7.10.0)
Requirement already satisfied: beautifulsoup4 in /share/apps/anaconda3/2024.
02/lib/python3.11/site-packages (from nbconvert) (4.12.2)
Requirement already satisfied: bleach!=5.0.0 in /share/apps/anaconda3/2024.0
2/lib/python3.11/site-packages (from nbconvert) (4.1.0)
Requirement already satisfied: defusedxml in /share/apps/anaconda3/2024.02/l
ib/python3.11/site-packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in /share/apps/anaconda3/2024.02/
lib/python3.11/site-packages (from nbconvert) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in /share/apps/anaconda3/20
24.02/lib/python3.11/site-packages (from nbconvert) (5.5.0)
Requirement already satisfied: jupyterlab-pygments in /share/apps/anaconda3/
2024.02/lib/python3.11/site-packages (from nbconvert) (0.1.2)
Requirement already satisfied: markupsafe>=2.0 in /share/apps/anaconda3/202
4.02/lib/python3.11/site-packages (from nbconvert) (2.1.3)
Requirement already satisfied: mistune<4,>=2.0.3 in /share/apps/anaconda3/20
24.02/lib/python3.11/site-packages (from nbconvert) (2.0.4)
Requirement already satisfied: nbclient>=0.5.0 in /share/apps/anaconda3/202
4.02/lib/python3.11/site-packages (from nbconvert) (0.8.0)
Requirement already satisfied: nbformat>=5.7 in /share/apps/anaconda3/2024.0
2/lib/python3.11/site-packages (from nbconvert) (5.9.2)
Requirement already satisfied: packaging in /share/apps/anaconda3/2024.02/li
b/python3.11/site-packages (from nbconvert) (23.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /share/apps/anaconda
3/2024.02/lib/python3.11/site-packages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in /share/apps/anaconda3/202
4.02/lib/python3.11/site-packages (from nbconvert) (2.15.1)
Requirement already satisfied: tinycss2 in /share/apps/anaconda3/2024.02/li
b/python3.11/site-packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.1 in /share/apps/anaconda3/2024.
02/lib/python3.11/site-packages (from nbconvert) (5.7.1)
Requirement already satisfied: six>=1.9.0 in /share/apps/anaconda3/2024.02/l
ib/python3.11/site-packages (from bleach!=5.0.0->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /share/apps/anaconda3/2024.0
2/lib/python3.11/site-packages (from bleach!=5.0.0->nbconvert) (0.5.1)
Requirement already satisfied: platformdirs>=2.5 in /share/apps/anaconda3/20
24.02/lib/python3.11/site-packages (from jupyter-core>=4.7->nbconvert) (3.1
0.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /share/apps/anacond
a3/2024.02/lib/python3.11/site-packages (from nbclient>=0.5.0->nbconvert)
(8.6.0)
Requirement already satisfied: fastjsonschema in /share/apps/anaconda3/2024.
02/lib/python3.11/site-packages (from nbformat>=5.7->nbconvert) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in /share/apps/anaconda3/202
4.02/lib/python3.11/site-packages (from nbformat>=5.7->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in /share/apps/anaconda3/2024.0
2/lib/python3.11/site-packages (from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: attrs>=22.2.0 in /share/apps/anaconda3/2024.0
2/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconve
rt) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /shar
e/apps/anaconda3/2024.02/lib/python3.11/site-packages (from jsonschema>=2.6-
>nbformat>=5.7->nbconvert) (2023.7.1)
```

```
Requirement already satisfied: referencing>=0.28.4 in /share/apps/anaconda3/
2024.02/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat>=5.7->n
bconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /share/apps/anaconda3/2024.
02/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconv
ert) (0.10.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /share/apps/anacond
a3/2024.02/lib/python3.11/site-packages (from jupyter-client>=6.1.12->nbclie
nt>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: pyzmq>=23.0 in /share/apps/anaconda3/2024.02/
lib/python3.11/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->
nbconvert) (25.1.2)
Requirement already satisfied: tornado>=6.2 in /share/apps/anaconda3/2024.0
2/lib/python3.11/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0
->nbconvert) (6.3.3)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]: