```
import torch
torch.manual_seed(17)

import numpy as np
from torchsummary import summary
from tqdm import tqdm
import matplotlib.pyplot as plt

from DatasetLoader import DatasetFetcher
from project_model import *
```

```
# if torch.backends.mps.is_available():
#     mps_device = torch.device("mps")
#     x = torch.ones(1, device=mps_device)
#     print (x)
# else:
#     print ("MPS device not found.")
```

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

```
cuda
```

```
# Fetching Dataset
df = DatasetFetcher(dataset="CIFAR10", batch_size=128)
df.addHorizontalFlipping()
#df.addVerticalFlipping()
df.addRandomCrop(size=32, padding=4)
#df.addAutoAugmentation()
#df.addHistogramEqualization()
df.addNormalizer()
#df.addGaussianNoise()
trainLoader, testLoader = df.getLoaders()
```

```
Initializing fetching CIFAR10 dataset using torchvision
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
```

```
# Get Model
#model = ResNet(BasicBlock, 32, 4, [4, 4, 4, 2], 10, bias=True)
model = project1_model()
model = model.to(device)
print(summary(model, input_size=(3, 32, 32)))
```

```
       BasicBlock-61            [-1, 128, 8, 8]               0
            Conv2d-62            [-1, 128, 8, 8]         147,584
       BatchNorm2d-63            [-1, 128, 8, 8]             256
            Conv2d-64            [-1, 128, 8, 8]         147,584
       BatchNorm2d-65            [-1, 128, 8, 8]             256
        BasicBlock-66            [-1, 128, 8, 8]               0
            Conv2d-67            [-1, 256, 4, 4]         295,168
       BatchNorm2d-68            [-1, 256, 4, 4]             512
            Conv2d-69            [-1, 256, 4, 4]         590,080
       BatchNorm2d-70            [-1, 256, 4, 4]             512
            Conv2d-71            [-1, 256, 4, 4]          33,024
       BatchNorm2d-72            [-1, 256, 4, 4]             512
        BasicBlock-73            [-1, 256, 4, 4]               0
            Conv2d-74            [-1, 256, 4, 4]         590,080
       BatchNorm2d-75            [-1, 256, 4, 4]             512
            Conv2d-76            [-1, 256, 4, 4]         590,080
       BatchNorm2d-77            [-1, 256, 4, 4]             512
        BasicBlock-78            [-1, 256, 4, 4]               0
            Linear-79                  [-1, 10]           2,570
================================================================
Total params: 3,576,842
Trainable params: 3,576,842
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 10.00
Params size (MB): 13.64
Estimated Total Size (MB): 23.66
----------------------------------------------------------------
None
```

```
EPOCHS= 100
globalBestAccuracy = 0.0
trainingLoss = []
testingLoss = []
trainingAccuracy = []
testingAccuracy = []
```

```
# Defining Loss Function, Learning Rate, Weight Decay, Optimizer)
lossFunction = torch.nn.CrossEntropyLoss(reduction='sum')
learningRate = 0.1
weightDecay = 0.0001
#optimizer = torch.optim.Adam(model.parameters(), lr=learningRate, weight_decay=weightDecay)
optimizer = torch.optim.Adagrad(model.parameters(), lr=learningRate, weight_decay=weightDecay)
#optimizer = torch.optim.Adadelta(model.parameters(), lr=learningRate, weight_decay=weightDecay)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, EPOCHS, eta_min=learningRate/10.0)
print(model.eval())
trainable_parameters = sum(p.numel() for p in model.parameters() if p.requires_grad)
print("Total Trainable Parameters : %s"%(trainable_parameters))
if trainable_parameters > 5*(10**6):
    raise Exception("Model not under budget!")
```

```
ResNet(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
    (3): BasicBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
        (shortcut): Sequential()
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential(
          (0): Conv2d(32, 64, kernel_size=(1, 1), stride=(2, 2))
          (1): 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))
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential()
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential()
      )
```

```
# Training
for i in tqdm(range(EPOCHS)):
    for phase in ['train', 'test']:
        if phase == "train":
            loader = trainLoader
            model.train()
            optimizer.zero_grad()
        else:
            loader = testLoader
            model.eval()
        runningLoss = 0.0
        runningCorrects = 0
        for images, labels in loader:
            images = images.to(device)
            labels = labels.to(device)
            output = model(images)
            loss = lossFunction(output, labels)
            predicted_labels = torch.argmax(output, dim=1)
            #runningLoss += loss.item()*images.size(0)
            runningLoss += loss.item()
            runningCorrects += torch.sum(predicted_labels == labels).float().item()
            if phase == "train":
                loss.backward()
                optimizer.step()
        epochLoss = runningLoss/len(loader.dataset)
        epochAccuracy = runningCorrects/len(loader.dataset)
        if phase == "train":
            scheduler.step()
            trainingLoss.append(epochLoss)
            trainingAccuracy.append(epochAccuracy)
        else:
            testingLoss.append(epochLoss)
            testingAccuracy.append(epochAccuracy)
            if epochAccuracy > globalBestAccuracy:
                globalBestAccuracy = epochAccuracy
                model.saveToDisk()
    print("Training Loss : %s, Testing Loss : %s, Training Accuracy : %s, Testing Accuracy : %s"\
          %(trainingLoss[-1], testingLoss[-1], trainingAccuracy[-1], testingAccuracy[-1]))
```

```
/it]Training Loss : 2.175165841369629, Testing Loss : 2.406909534263611, Training Accuracy : 0.20754, Testing Accuracy : 0.24
4s/it]Training Loss : 1.9478375146484375, Testing Loss : 1.8775852556228638, Training Accuracy : 0.26596, Testing Accuracy :
4s/it]Training Loss : 1.843938726196289, Testing Loss : 1.81078774394989, Training Accuracy : 0.30692, Testing Accuracy : 0.3
4s/it]Training Loss : 1.7810005841064454, Testing Loss : 1.7035266538619995, Training Accuracy : 0.33272, Testing Accuracy :
6s/it]Training Loss : 1.7421606359863282, Testing Loss : 1.6967677848815919, Training Accuracy : 0.35132, Testing Accuracy :
/it]  Training Loss : 1.6967336584472656, Testing Loss : 1.642623657989502, Training Accuracy : 0.37014, Testing Accuracy : 0
/it]Training Loss : 1.6709186674499512, Testing Loss : 1.6596612241744995, Training Accuracy : 0.3789, Testing Accuracy : 0.3
/it]Training Loss : 1.6471936547851562, Testing Loss : 1.5868560096740723, Training Accuracy : 0.39248, Testing Accuracy : 0.
/it]Training Loss : 1.6135773054504394, Testing Loss : 1.5516728332519532, Training Accuracy : 0.40478, Testing Accuracy : 0.
s/it]Training Loss : 1.5836825, Testing Loss : 1.5309457374572755, Training Accuracy : 0.41242, Testing Accuracy : 0.4405
s/it]Training Loss : 1.553962603149414, Testing Loss : 1.4984964988708496, Training Accuracy : 0.4294, Testing Accuracy : 0.4
s/it]Training Loss : 1.5174052783203125, Testing Loss : 1.4585836734771729, Training Accuracy : 0.43876, Testing Accuracy : 0
s/it]Training Loss : 1.4944309078979492, Testing Loss : 1.4494610462188722, Training Accuracy : 0.44732, Testing Accuracy : 0
```

```
s/it]Training Loss : 1.4682401776123046, Testing Loss : 1.4298958913803101, Training Accuracy : 0.45662, Testing Accuracy : 0
s/it]Training Loss : 1.4486421473693847, Testing Loss : 1.422221272277832, Training Accuracy : 0.46658, Testing Accuracy : 0.
s/it]Training Loss : 1.4473827821350098, Testing Loss : 1.4325064603805542, Training Accuracy : 0.46756, Testing Accuracy : 0
s/it]Training Loss : 1.4234435266113281, Testing Loss : 1.390982437133789, Training Accuracy : 0.4745, Testing Accuracy : 0.4
s/it]Training Loss : 1.385718712310791, Testing Loss : 1.3577065141677855, Training Accuracy : 0.4929, Testing Accuracy : 0.5
s/it]Training Loss : 1.3488667247009278, Testing Loss : 1.3105208967208863, Training Accuracy : 0.50538, Testing Accuracy : 0
s/it]Training Loss : 1.3272379940795898, Testing Loss : 1.3256290180206298, Training Accuracy : 0.51546, Testing Accuracy : 0
s/it]Training Loss : 1.3105348844909668, Testing Loss : 1.2576984516143799, Training Accuracy : 0.52392, Testing Accuracy : 0
s/it]Training Loss : 1.2788981851196288, Testing Loss : 1.2944715857505797, Training Accuracy : 0.53482, Testing Accuracy : 0
s/it]Training Loss : 1.2574534729003906, Testing Loss : 1.22915731010437, Training Accuracy : 0.54466, Testing Accuracy : 0.5
s/it]Training Loss : 1.232506286468506, Testing Loss : 1.2035089353561401, Training Accuracy : 0.55458, Testing Accuracy : 0.
s/it]Training Loss : 1.2071748132324218, Testing Loss : 1.1938717779159547, Training Accuracy : 0.56524, Testing Accuracy : 0
s/it]Training Loss : 1.182650696258545, Testing Loss : 1.1349238962173462, Training Accuracy : 0.5742, Testing Accuracy : 0.5
s/it]Training Loss : 1.1554034294128417, Testing Loss : 1.1701407571792604, Training Accuracy : 0.58728, Testing Accuracy : 0
s/it]Training Loss : 1.1391474661254883, Testing Loss : 1.1227729215621949, Training Accuracy : 0.5913, Testing Accuracy : 0.
s/it]Training Loss : 1.1148492536926269, Testing Loss : 1.0965403210639955, Training Accuracy : 0.60044, Testing Accuracy : 0
s/it]Training Loss : 1.0818005854797363, Testing Loss : 1.09345222492218, Training Accuracy : 0.61178, Testing Accuracy : 0.6
s/it]Training Loss : 1.0828966236877442, Testing Loss : 1.1250913299560548, Training Accuracy : 0.6124, Testing Accuracy : 0.
s/it]Training Loss : 1.0596564138793945, Testing Loss : 1.0488066653251649, Training Accuracy : 0.62052, Testing Accuracy : 0
s/it]Training Loss : 1.0330932772827148, Testing Loss : 1.0168566140174866, Training Accuracy : 0.6296, Testing Accuracy : 0.
s/it]Training Loss : 1.0023919076538086, Testing Loss : 0.9898117404937744, Training Accuracy : 0.64306, Testing Accuracy : 0
s/it]Training Loss : 0.9797069189453125, Testing Loss : 1.0066391167640687, Training Accuracy : 0.65006, Testing Accuracy : 0
s/it]Training Loss : 0.9716653312683106, Testing Loss : 0.9846182690620422, Training Accuracy : 0.65292, Testing Accuracy : 0
s/it]Training Loss : 0.9663718643188477, Testing Loss : 0.978037689781189, Training Accuracy : 0.65594, Testing Accuracy : 0.
s/it]Training Loss : 0.9566370715332031, Testing Loss : 0.9756511347770691, Training Accuracy : 0.65938, Testing Accuracy : 0
s/it]Training Loss : 0.9317118572998047, Testing Loss : 0.9613517046928406, Training Accuracy : 0.67174, Testing Accuracy : 0
s/it]Training Loss : 0.9205063075256348, Testing Loss : 0.9412313754081726, Training Accuracy : 0.67336, Testing Accuracy : 0
s/it]Training Loss : 0.8971576127624512, Testing Loss : 0.9360193239212036, Training Accuracy : 0.6813, Testing Accuracy : 0.
s/it]Training Loss : 0.8805332716369629, Testing Loss : 0.9223612445831298, Training Accuracy : 0.68724, Testing Accuracy : 0
s/it]Training Loss : 0.8694712023925781, Testing Loss : 0.9277423460006714, Training Accuracy : 0.69334, Testing Accuracy : 0
s/it]Training Loss : 0.8558593743133545, Testing Loss : 0.9031948948860169, Training Accuracy : 0.69516, Testing Accuracy : 0
s/it]Training Loss : 0.8523868218994141, Testing Loss : 0.8786452639579773, Training Accuracy : 0.6981, Testing Accuracy : 0.
s/it]Training Loss : 0.835121849899292, Testing Loss : 0.8593402626991272, Training Accuracy : 0.70106, Testing Accuracy : 0.
s/it]Training Loss : 0.8184041098022461, Testing Loss : 0.866538259601593, Training Accuracy : 0.71156, Testing Accuracy : 0.
s/it]Training Loss : 0.8244605818176269, Testing Loss : 0.8432361203193665, Training Accuracy : 0.70818, Testing Accuracy : 0
s/it]Training Loss : 0.7997285914611817, Testing Loss : 0.8391276433944702, Training Accuracy : 0.71816, Testing Accuracy : 0
s/it]Training Loss : 0.7775847235870361, Testing Loss : 0.8100645894050598, Training Accuracy : 0.72454, Testing Accuracy : 0
s/it]Training Loss : 0.7665461134338379, Testing Loss : 0.8251302044868469, Training Accuracy : 0.72976, Testing Accuracy : 0
s/it]Training Loss : 0.7496114512634278, Testing Loss : 0.7949952085494996, Training Accuracy : 0.73366, Testing Accuracy : 0
s/it]Training Loss : 0.744664236831665, Testing Loss : 0.8023356796264648, Training Accuracy : 0.73436, Testing Accuracy : 0.
s/it]Training Loss : 0.7385432836914062, Testing Loss : 0.7886579656600952, Training Accuracy : 0.73818, Testing Accuracy : 0
s/it]Training Loss : 0.7389554113006592, Testing Loss : 0.8155789810180664, Training Accuracy : 0.73804, Testing Accuracy : 0
s/it]Training Loss : 0.7291836672210693, Testing Loss : 0.7784889444351196, Training Accuracy : 0.74428, Testing Accuracy : 0
s/it]Training Loss : 0.7117158071136475, Testing Loss : 0.7914186009407044, Training Accuracy : 0.74728, Testing Accuracy : 0
s/it]Training Loss : 0.7109982388305665, Testing Loss : 0.7535451029777527, Training Accuracy : 0.75082, Testing Accuracy : 0
```

```
print("Maximum Testing Accuracy Achieved: %s"%(max(testingAccuracy)))
xmax = np.argmax(testingAccuracy)
ymax = max(testingAccuracy)
```

```
Maximum Testing Accuracy Achieved: 0.7898
```

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
n = len(trainingLoss)
ax1.plot(range(n), trainingLoss, '-', linewidth='3', label='Train Error')
ax1.plot(range(n), testingLoss, '-', linewidth='3', label='Test Error')
ax2.plot(range(n), trainingAccuracy, '-', linewidth='3', label='Train Accuracy')
ax2.plot(range(n), testingAccuracy, '-', linewidth='3', label='Test Acuracy')
ax2.annotate('max accuracy = %s'%(ymax), xy=(xmax, ymax), xytext=(xmax, ymax+0.15), arrowprops=dict(facecolor='black', shrink=0.0
ax1.grid(True)
ax2.grid(True)
ax1.legend()
ax2.legend()
f.savefig("./trainTestCurve.png")
```