```
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
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:03<00:00, 48348481.17it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
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)))
```

```
            Conv2d-59            [-1, 128, 8, 8]         147,584
       BatchNorm2d-60            [-1, 128, 8, 8]             256
        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!")
```

```
      (3): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential()
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2))
          (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))
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential()
      )
    )
    (linear): Linear(in_features=256, out_features=10, bias=True)
  )
  Total Trainable Parameters : 3576842
```

```
# 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]))
```

```
7.02s/it]Training Loss : 6.2935484423828125, Testing Loss : 8.595463990783692, Training Accuracy : 0.10016, Testing Accuracy
7.15s/it]Training Loss : 7.845943788452148, Testing Loss : 32.153229711914065, Training Accuracy : 0.10262, Testing Accuracy
7.39s/it]Training Loss : 9.06403954650879, Testing Loss : 7.7073091110229495, Training Accuracy : 0.09892, Testing Accuracy :
7.28s/it]Training Loss : 5.896392215576172, Testing Loss : 9.769394424438477, Training Accuracy : 0.10052, Testing Accuracy :
7.59s/it]Training Loss : 6.045679028320312, Testing Loss : 6.192060710906983, Training Accuracy : 0.10004, Testing Accuracy :
7.30s/it]Training Loss : 6.470145396728515, Testing Loss : 14.199798805236817, Training Accuracy : 0.09842, Testing Accuracy
7.20s/it]Training Loss : 7.7784174890136715, Testing Loss : 7.739736657714844, Training Accuracy : 0.10098, Testing Accuracy
7.27s/it]Training Loss : 7.199571335449218, Testing Loss : 9.77441015625, Training Accuracy : 0.0986, Testing Accuracy : 0.1
7.13s/it]Training Loss : 7.152112897338867, Testing Loss : 14.994243103027344, Training Accuracy : 0.0997, Testing Accuracy :
7.17s/it]Training Loss : 4.978037475585937, Testing Loss : 6.079327964782715, Training Accuracy : 0.1, Testing Accuracy : 0.1
6.95s/it]Training Loss : 4.935128229370117, Testing Loss : 5.357398912811279, Training Accuracy : 0.09972, Testing Accuracy :
7.00s/it]Training Loss : 4.56347008972168, Testing Loss : 300.47023364257814, Training Accuracy : 0.1002, Testing Accuracy :
6.93s/it]Training Loss : 4.447053453369141, Testing Loss : 3.707677317047119, Training Accuracy : 0.09912, Testing Accuracy :
6.75s/it]Training Loss : 3.558818461303711, Testing Loss : 4.1941743637084965, Training Accuracy : 0.09952, Testing Accuracy
6.85s/it]Training Loss : 3.8408408422851563, Testing Loss : 3.7191454723358155, Training Accuracy : 0.10074, Testing Accuracy
6.93s/it]Training Loss : 3.408944842529297, Testing Loss : 4.701116198730468, Training Accuracy : 0.098, Testing Accuracy : 0
6.95s/it]Training Loss : 4.360525975341797, Testing Loss : 3.2969082534790037, Training Accuracy : 0.09924, Testing Accuracy
6.93s/it]Training Loss : 3.3018694677734377, Testing Loss : 297.92037016601563, Training Accuracy : 0.0999, Testing Accuracy
6.74s/it]Training Loss : 3.610156971435547, Testing Loss : 9156.805728125, Training Accuracy : 0.09796, Testing Accuracy : 0.
6.90s/it]Training Loss : 4.1080670208740235, Testing Loss : 43.93257971801758, Training Accuracy : 0.10226, Testing Accuracy
6.70s/it]Training Loss : 3.4214746380615235, Testing Loss : 3.085892850112915, Training Accuracy : 0.10192, Testing Accuracy
6.89s/it]Training Loss : 3.050325185546875, Testing Loss : 3.0605140602111818, Training Accuracy : 0.10092, Testing Accuracy
6.89s/it]Training Loss : 2.859252075805664, Testing Loss : 144.23668212890624, Training Accuracy : 0.09828, Testing Accuracy
6.80s/it]Training Loss : 3.244547919616699, Testing Loss : 3.1283012630462648, Training Accuracy : 0.09782, Testing Accuracy
6.94s/it]Training Loss : 2.9990178787231447, Testing Loss : 2.820087843322754, Training Accuracy : 0.10122, Testing Accuracy
6.84s/it]Training Loss : 2.7419374157714844, Testing Loss : 3.234247675704956, Training Accuracy : 0.09786, Testing Accuracy
7.05s/it]Training Loss : 3.1642532406616213, Testing Loss : 25.964326080322266, Training Accuracy : 0.10052, Testing Accuracy
7.10s/it]Training Loss : 2.8148546466064452, Testing Loss : 2.5686869968414308, Training Accuracy : 0.09856, Testing Accuracy
6.93s/it]Training Loss : 2.5158463439941405, Testing Loss : 2.904122500228882, Training Accuracy : 0.10024, Testing Accuracy
7.14s/it]Training Loss : 3.0147257830810545, Testing Loss : 60.358043493652346, Training Accuracy : 0.09888, Testing Accuracy
6.99s/it]Training Loss : 2.6809927825927735, Testing Loss : 2.7926663669586183, Training Accuracy : 0.10088, Testing Accuracy
7.15s/it]Training Loss : 2.939789956359863, Testing Loss : 252.19242802734374, Training Accuracy : 0.09958, Testing Accuracy
7.06s/it]Training Loss : 2.849639326477051, Testing Loss : 2.8334620071411134, Training Accuracy : 0.10034, Testing Accuracy
6.94s/it]Training Loss : 2.669753287963867, Testing Loss : 2.7095908378601075, Training Accuracy : 0.09952, Testing Accuracy
6.88s/it]Training Loss : 2.6440085906982422, Testing Loss : 2.49813028755188, Training Accuracy : 0.09938, Testing Accuracy :
6.76s/it]Training Loss : 2.435522263793945, Testing Loss : 2.463446297836304, Training Accuracy : 0.09696, Testing Accuracy :
6.79s/it]Training Loss : 2.474990462036133, Testing Loss : 2.550361374282837, Training Accuracy : 0.10088, Testing Accuracy :
6.66s/it]Training Loss : 7.303676997070313, Testing Loss : 55.99706719970703, Training Accuracy : 0.10276, Testing Accuracy :
7.08s/it]Training Loss : 2.664364291687012, Testing Loss : 3.0600454498291016, Training Accuracy : 0.10136, Testing Accuracy
6.84s/it]Training Loss : 2.7894586935424805, Testing Loss : 5.107293126296997, Training Accuracy : 0.10166, Testing Accuracy
6.87s/it]Training Loss : 2.4923036560058596, Testing Loss : 2.4963308403015136, Training Accuracy : 0.09948, Testing Accuracy
6.95s/it]Training Loss : 2.3951123275756836, Testing Loss : 2.373703150177002, Training Accuracy : 0.10174, Testing Accuracy
6.72s/it]Training Loss : 2.3899280883789062, Testing Loss : 2.4489249584198, Training Accuracy : 0.09906, Testing Accuracy :
 37.02s/it]Training Loss : 2.447885647583008, Testing Loss : 2.5533477127075197, Training Accuracy : 0.10268, Testing Accurac
 36.87s/it]Training Loss : 2.494013700256348, Testing Loss : 85.07906862792969, Training Accuracy : 0.09884, Testing Accuracy
, 36.93s/it]Training Loss : 2.4765010595703125, Testing Loss : 339.93812893066405, Training Accuracy : 0.09884, Testing Accur
```

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

```
 Maximum Testing Accuracy Achieved: 0.1616
```

```
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")
```

max accuracy = 0.1616