Target:

```
-- Our previous model has 6.3M parameters. I want to reduce these parameters here. I want to make them < 10K
```

Results:

```
-- Parameters: 9,907
-- Best Training Accuracy: 99.99
-- Best Test Accuracy: 99.05
```

Analysis:

```
-- The accuracy has dropped a little (from 99.61 to 99.05) after reducing the number of parameters
-- traing and test accuracy are increasing with epochs. So we are in the right path. We might improve results by training it for few
```

Import libraries

```
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
```

Data Transformations (without normalization)

```
# Train Phase transformations
train_transforms = transforms.Compose([
                                      transforms.ToTensor()
                                       ])

# Test Phase transformations
test_transforms = transforms.Compose([
                                      transforms.ToTensor()
                                       ])
```

Dataset and Creating Train/Test Split (without normalization)

```
train = datasets.MNIST('./data', train=True, download=True, transform=train_transforms)
test = datasets.MNIST('./data', train=False, download=True, transform=test_transforms)
```

Dataloader Arguments & Test/Train Dataloaders (without normalization)

```
SEED = 1

# CUDA?
cuda = torch.cuda.is_available()
print("CUDA Available?", cuda)

# For reproducibility
torch.manual_seed(SEED)

if cuda:
    torch.cuda.manual_seed(SEED)

# dataloader arguments - something you'll fetch these from cmdprmt
dataloader_args = dict(shuffle=True, batch_size=128, num_workers=4, pin_memory=True) if cuda else dict(shuffle=True, batch

# train dataloader
train_loader = torch.utils.data.DataLoader(train, **dataloader_args)

# test dataloader
test_loader = torch.utils.data.DataLoader(test, **dataloader_args)
```

```
CUDA Available? True
/usr/local/lib/python3.8/dist-packages/torch/utils/data/dataloader.py:554: UserWarning: This DataLoader will create 4
  warnings.warn(_create_warning_msg(
```

Getting data statistics (without normalization)

We will use the mean and standard deviation that we get from code below to normalize the data

```
import numpy as np

train_data = train.train_data
train_data = train.transform(train_data.numpy())

print('[Train]')
print(' - Numpy Shape:', train.train_data.cpu().numpy().shape)
print(' - Tensor Shape:', train.train_data.size())
print(' - min:', torch.min(train_data))
print(' - max:', torch.max(train_data))
print(' - mean:', torch.mean(train_data))
print(' - std:', torch.std(train_data))
print(' - var:', torch.var(train_data))

dataiter = iter(train_loader)
images, labels = next(dataiter)

print(images.shape)
print(labels.shape)

# Let's visualize some of the images
%matplotlib inline
import matplotlib.pyplot as plt

plt.imshow(images[0].numpy().squeeze(), cmap='gray_r')
```

Data Transformations (with normalization)

```
# Train Phase transformations
train_transforms = transforms.Compose([
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.1307,), (0.3081,))
                                       ])

# Test Phase transformations
test_transforms = transforms.Compose([
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.1307,), (0.3081,))
                                       ])
```

Dataset and Creating Train/Test Split (with normalization)

```
train = datasets.MNIST('./data', train=True, download=True, transform=train_transforms)
test = datasets.MNIST('./data', train=False, download=True, transform=test_transforms)
```

Dataloader Arguments & Test/Train Dataloaders (with normalization)

```
SEED = 1

# CUDA?
cuda = torch.cuda.is_available()
print("CUDA Available?", cuda)

# For reproducibility
torch.manual_seed(SEED)

if cuda:
    torch.cuda.manual_seed(SEED)

# dataloader arguments - something you'll fetch these from cmdprmt
dataloader_args = dict(shuffle=True, batch_size=128, num_workers=4, pin_memory=True) if cuda else dict(shuffle=True, batch

# train dataloader
train_loader = torch.utils.data.DataLoader(train, **dataloader_args)

# test dataloader
test_loader = torch.utils.data.DataLoader(test, **dataloader_args)
```

```
    CUDA Available? True
```

Getting data statistics (with normalization)

We will use the mean and standard deviation that we get from code below to normalize the data

```
import numpy as np

train_data = train.train_data
train_data = train.transform(train_data.numpy())

print('[Train]')
print(' - Numpy Shape:', train.train_data.cpu().numpy().shape)
print(' - Tensor Shape:', train.train_data.size())
print(' - min:', torch.min(train_data))
print(' - max:', torch.max(train_data))
```

```
print(' - mean:', torch.mean(train_data))
print(' - std:', torch.std(train_data))
print(' - var:', torch.var(train_data))

dataiter = iter(train_loader)
images, labels = next(dataiter)

print(images.shape)
print(labels.shape)

# Let's visualize some of the images
%matplotlib inline
import matplotlib.pyplot as plt

plt.imshow(images[0].numpy().squeeze(), cmap='gray_r')
```

Model

```
dropout_rate = 0.05
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()

    #input block
    self.convblock1 = nn.Sequential(nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(16),
                                    nn.ReLU()) #R_in = 1, C_in = 28, K = 3, P = 1, S = 1, J_in = 1, J_out = 1, R_out = R_i

    #conv block 1
    self.convblock2 = nn.Sequential(nn.Conv2d(in_channels = 16, out_channels = 12, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(12),
                                    nn.ReLU()) #R_in = 3, C_in = 28, K = 3, P = 1, S = 1, J_in = 1, J_out = 1, R_out = R_i

    #conv block 2
    self.convblock3 = nn.Sequential(nn.Conv2d(in_channels = 12, out_channels = 16, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(16),
                                    nn.ReLU()) #R_in = 5, C_in = 28, K = 3, P = 1, S = 1, J_in = 1, J_out = 1, R_out = R_i

    #transition block1
    self.convblock4 = nn.Sequential(nn.Conv2d(in_channels = 16, out_channels = 12, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(12),
                                    nn.ReLU()) #R_in = 7, C_in = 28, K = 3, P = 1, S = 1, J_in = 1, J_out = 1, R_out = R_i

    self.pool1 = nn.MaxPool2d(2, 2) #R_in = 9, C_in = 28, K = 2, P = 0, S = 2, J_in = 1, J_out = 2, R_out = R_in + (K-1)*J

    #conv block 3
    self.convblock5 = nn.Sequential(nn.Conv2d(in_channels = 12, out_channels = 15, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(15),
                                    nn.ReLU()) #R_in = 9, C_in = 14, K = 3, P = 1, S = 1, J_in = 2, J_out = 2, R_out = R_i

    #conv block 4
    self.convblock6 = nn.Sequential(nn.Conv2d(in_channels = 15, out_channels = 12, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(12),
                                    nn.ReLU()) #R_in = 13, C_in = 14, K = 3, P = 1, S = 1, J_in = 2, J_out = 2, R_out = R_
    #gap layer
```

```
    self.gap = nn.Sequential(
            nn.AvgPool2d(kernel_size=4))  #R_in = 17, C_in = 14, K = 4, P = 1, S = 1, J_in = 2, J_out = 2, R_out = R_in +

    #output block
    self.convblock7 = nn.Sequential(nn.Conv2d(in_channels = 12, out_channels = 10, kernel_size = 3, padding = 0)) #R_in =

  def forward(self, x):
    x = self.convblock1(x)
    x = self.convblock2(x)
    x = self.convblock3(x)
    x = self.convblock4(x)
    x = self.pool1(x)
    x = self.convblock5(x)
    x = self.convblock6(x)
    x = self.gap(x)
    x = self.convblock7(x)
    x = x.view(-1, 10)
    return F.log_softmax(x, dim=-1)
```

Model parameters

```
!pip install torchsummary
from torchsummary import summary

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

model = Net().to(device)
summary(model, input_size = (1, 28, 28))
```

```
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: torchsummary in /usr/local/lib/python3.8/dist-packages (1.5.1)
    ----------------------------------------------------------------
            Layer (type)               Output Shape         Param #
    ================================================================
                Conv2d-1           [-1, 16, 28, 28]             160
           BatchNorm2d-2           [-1, 16, 28, 28]              32
                  ReLU-3           [-1, 16, 28, 28]               0
                Conv2d-4           [-1, 12, 28, 28]           1,740
           BatchNorm2d-5           [-1, 12, 28, 28]              24
                  ReLU-6           [-1, 12, 28, 28]               0
                Conv2d-7           [-1, 16, 28, 28]           1,744
           BatchNorm2d-8           [-1, 16, 28, 28]              32
                  ReLU-9           [-1, 16, 28, 28]               0
               Conv2d-10           [-1, 12, 28, 28]           1,740
          BatchNorm2d-11           [-1, 12, 28, 28]              24
                 ReLU-12           [-1, 12, 28, 28]               0
            MaxPool2d-13           [-1, 12, 14, 14]               0
               Conv2d-14           [-1, 15, 14, 14]           1,635
          BatchNorm2d-15           [-1, 15, 14, 14]              30
                 ReLU-16           [-1, 15, 14, 14]               0
               Conv2d-17           [-1, 12, 14, 14]           1,632
          BatchNorm2d-18           [-1, 12, 14, 14]              24
                 ReLU-19           [-1, 12, 14, 14]               0
            AvgPool2d-20             [-1, 12, 3, 3]               0
               Conv2d-21             [-1, 10, 1, 1]           1,090
    ================================================================
    Total params: 9,907
    Trainable params: 9,907
    Non-trainable params: 0
    ----------------------------------------------------------------
    Input size (MB): 0.00
    Forward/backward pass size (MB): 1.14
    Params size (MB): 0.04
    Estimated Total Size (MB): 1.19
    ----------------------------------------------------------------
```

Training and Testing

```
from tqdm import tqdm

train_losses = []
test_losses = []
train_acc = []
test_acc = []

def train(model, device, train_loader, optimizer, epoch):
  model.train()
  pbar = tqdm(train_loader)
  correct = 0
```

```
    processed = 0
    for batch_idx, (data, target) in enumerate(pbar):
      # get samples
      data, target = data.to(device), target.to(device)

      # Init
      optimizer.zero_grad()
      # In PyTorch, we need to set the gradients to zero before starting to do backpropragation because PyTorch accumulates
      # Because of this, when you start your training loop, ideally you should zero out the gradients so that you do the par

      # Predict
      y_pred = model(data)

      # Calculate loss
      loss = F.nll_loss(y_pred, target)
      train_losses.append(loss)

      # Backpropagation
      loss.backward()
      optimizer.step()

      # Update pbar-tqdm

      pred = y_pred.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
      correct += pred.eq(target.view_as(pred)).sum().item()
      processed += len(data)

      pbar.set_description(desc= f'Loss={loss.item()} Batch_id={batch_idx} Accuracy={100*correct/processed:0.2f}')
      train_acc.append(100*correct/processed)

  def test(model, device, test_loader):
      model.eval()
      test_loss = 0
      correct = 0
      with torch.no_grad():
          for data, target in test_loader:
              data, target = data.to(device), target.to(device)
              output = model(data)
              test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
              pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
              correct += pred.eq(target.view_as(pred)).sum().item()

      test_loss /= len(test_loader.dataset)
      test_losses.append(test_loss)

      print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
          test_loss, correct, len(test_loader.dataset),
          100. * correct / len(test_loader.dataset)))

      test_acc.append(100. * correct / len(test_loader.dataset))


  from torch.optim.lr_scheduler import StepLR

  model =  Net().to(device)
  optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
  # scheduler = StepLR(optimizer, step_size=6, gamma=0.1)



  EPOCHS = 15
  for epoch in range(EPOCHS):
      print("EPOCH:", epoch)
      train(model, device, train_loader, optimizer, epoch)
      # scheduler.step()
      test(model, device, test_loader)
```

```
EPOCH: 0
Loss=0.12683193385601044 Batch_id=468 Accuracy=92.11: 100%|██████████| 469/469 [00:20<00:00, 22.48it/s]

Test set: Average loss: 0.0998, Accuracy: 9706/10000 (97.06%)

EPOCH: 1
Loss=0.09563467651605606 Batch_id=468 Accuracy=98.14: 100%|██████████| 469/469 [00:17<00:00, 27.53it/s]

Test set: Average loss: 0.0539, Accuracy: 9832/10000 (98.32%)

EPOCH: 2
Loss=0.039049576967954636 Batch_id=468 Accuracy=98.51: 100%|██████████| 469/469 [00:14<00:00, 31.98it/s]

Test set: Average loss: 0.0937, Accuracy: 9731/10000 (97.31%)

EPOCH: 3
```

```
Loss=0.0362037755548954 Batch_id=468 Accuracy=98.84: 100%|██████████| 469/469 [00:14<00:00, 32.28it/s]

Test set: Average loss: 0.0563, Accuracy: 9814/10000 (98.14%)

EPOCH: 4
Loss=0.025478513911366463 Batch_id=468 Accuracy=98.91: 100%|██████████| 469/469 [00:14<00:00, 32.17it/s]

Test set: Average loss: 0.0471, Accuracy: 9846/10000 (98.46%)

EPOCH: 5
Loss=0.03193654492497444 Batch_id=468 Accuracy=99.06: 100%|██████████| 469/469 [00:14<00:00, 32.13it/s]

Test set: Average loss: 0.0390, Accuracy: 9867/10000 (98.67%)

EPOCH: 6
Loss=0.021154969930648804 Batch_id=468 Accuracy=99.14: 100%|██████████| 469/469 [00:15<00:00, 31.19it/s]

Test set: Average loss: 0.0325, Accuracy: 9906/10000 (99.06%)

EPOCH: 7
Loss=0.05115535482764244 Batch_id=468 Accuracy=99.24: 100%|██████████| 469/469 [00:14<00:00, 32.04it/s]

Test set: Average loss: 0.0365, Accuracy: 9891/10000 (98.91%)

EPOCH: 8
Loss=0.0016950649442151189 Batch_id=468 Accuracy=99.26: 100%|██████████| 469/469 [00:15<00:00, 30.91it/s]

Test set: Average loss: 0.0279, Accuracy: 9912/10000 (99.12%)

EPOCH: 9
Loss=0.005443142727017403 Batch_id=468 Accuracy=99.33: 100%|██████████| 469/469 [00:14<00:00, 31.89it/s]

Test set: Average loss: 0.0300, Accuracy: 9894/10000 (98.94%)

EPOCH: 10
Loss=0.011261711828410625 Batch_id=468 Accuracy=99.41: 100%|██████████| 469/469 [00:14<00:00, 32.27it/s]

Test set: Average loss: 0.0333, Accuracy: 9888/10000 (98.88%)

EPOCH: 11
Loss=0.00586252985522151 Batch_id=468 Accuracy=99.41: 100%|██████████| 469/469 [00:16<00:00, 29.11it/s]
```

```
train_losses = [i.item() for i in train_losses]
```

```
%matplotlib inline
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2,2,figsize=(15,10))
axs[0, 0].plot(train_losses)
axs[0, 0].set_title("Training Loss")
axs[1, 0].plot(train_acc[4000:])
axs[1, 0].set_title("Training Accuracy")
axs[0, 1].plot(test_losses)
axs[0, 1].set_title("Test Loss")
axs[1, 1].plot(test_acc)
axs[1, 1].set_title("Test Accuracy")
```