Target:

```
-- Get the set-up right
-- Set Transforms
-- Set Data Loader
-- Set Basic Working Code
-- Set Basic Training  & Test Loop
-- Use batch normalisation
```

Results:

```
-- Parameters: 6,383,818
-- Best Training Accuracy: 99.97
-- Best Test Accuracy: 99.56
```

Analysis:

```
-- The accuracy is really good.
-- The model is starting to overfitting in last few eopchs as the test accuracy is decreasing along with the training accuracy
-- model is really heavy. 6.3M parameters are really heavy
```

Import libraries

```
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.functional as F
```


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)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.g
100%                                          9912422/9912422 [00:00<00:00, 18324015.82it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.g
```

Dataloader Arguments & Test/Train Dataloaders (without normalization)

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
```

```
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(
```

The shortcut 'Print notebook' is disabled when a code cell output iframe is active. Use the escape key to leave the iframe and enter the shortcut again.

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')
```

```
/usr/local/lib/python3.8/dist-packages/torchvision/datasets/mnist.py:75: UserWarning: train_data has been renamed dat
  warnings.warn("train_data has been renamed data")
[Train]
 - Numpy Shape: (60000, 28, 28)
 - Tensor Shape: torch.Size([60000, 28, 28])
 - min: tensor(0.)
 - max: tensor(1.)
 - mean: tensor(0.1307)
 - std: tensor(0.3081)
 - var: tensor(0.0949)
torch.Size([128, 1, 28, 28])
torch.Size([128])
<matplotlib.image.AxesImage at 0x7f4d91edec70>
```

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)

The shortcut 'Print notebook' is disabled when a code cell output iframe is active. Use the escape key to leave the iframe and enter the shortcut again.

```
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
```

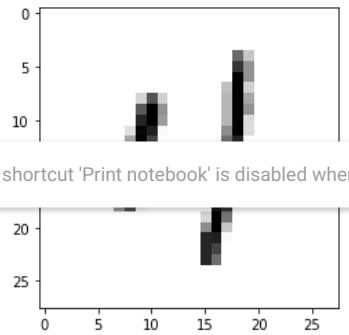
Getting data statistics (with normalization)

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')
```

```
[Train]
 - Numpy Shape: (60000, 28, 28)
 - Tensor Shape: torch.Size([60000, 28, 28])
 - min: tensor(-0.4242)
 - max: tensor(2.8215)
 - mean: tensor(-0.0001)
 - std: tensor(1.0000)
 - var: tensor(1.0001)
torch.Size([128, 1, 28, 28])
torch.Size([128])
<matplotlib.image.AxesImage at 0x7f4d910474f0>
```



Model

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

    #input block
    self.convblock1 = nn.Sequential(nn.Conv2d(in_channels = 1, out_channels = 32, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(32),
                                    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 = 32, out_channels = 64, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(64),
                                    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 = 64, out_channels = 128, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(128),
                                    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 = 128, out_channels = 256, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(256),
                                    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 = 256, out_channels = 512, kernel_size = 3, padding = 1),
                                    nn.BatchNorm2d(512),
```

```
                                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 = 512, out_channels = 1024, kernel_size = 3, padding = 1),
                                nn.BatchNorm2d(1024),
                                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 = 1024, out_channels = 10, kernel_size = 3, padding = 0))
                      #R_in = 23, C_in = 14, K = 3, P = 0, S = 1, J_in = 2, J_out = 2, R_out = R_in + (K-1)*J_in =  23+

  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)
```



```
Requirement already satisfied: torchsummary in /usr/local/lib/python3.8/dist-packages (1.5.1)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 28]             320
       BatchNorm2d-2           [-1, 32, 28, 28]              64
              ReLU-3           [-1, 32, 28, 28]               0
            Conv2d-4           [-1, 64, 28, 28]          18,496
       BatchNorm2d-5           [-1, 64, 28, 28]             128
              ReLU-6           [-1, 64, 28, 28]               0
            Conv2d-7          [-1, 128, 28, 28]          73,856
       BatchNorm2d-8          [-1, 128, 28, 28]             256
              ReLU-9          [-1, 128, 28, 28]               0
           Conv2d-10          [-1, 256, 28, 28]         295,168
      BatchNorm2d-11          [-1, 256, 28, 28]             512
             ReLU-12          [-1, 256, 28, 28]               0
        MaxPool2d-13          [-1, 256, 14, 14]               0
           Conv2d-14          [-1, 512, 14, 14]       1,180,160
      BatchNorm2d-15          [-1, 512, 14, 14]           1,024
             ReLU-16          [-1, 512, 14, 14]               0
           Conv2d-17         [-1, 1024, 14, 14]       4,719,616
      BatchNorm2d-18         [-1, 1024, 14, 14]           2,048
             ReLU-19         [-1, 1024, 14, 14]               0
        AvgPool2d-20           [-1, 1024, 3, 3]               0
           Conv2d-21             [-1, 10, 1, 1]          92,170
================================================================
Total params: 6,383,818
Trainable params: 6,383,818
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 15.96
Params size (MB): 24.35
Estimated Total Size (MB): 40.31
----------------------------------------------------------------
```

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)
```



```
    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=0.04363084211945534 Batch_id=468 Accuracy=96.39: 100%|██████████| 469/469 [01:10<00:00,  6.63it/s]

Test set: Average loss: 0.0355, Accuracy: 9881/10000 (98.81%)

EPOCH: 1
Loss=0.014005164615809917 Batch_id=468 Accuracy=98.99: 100%|██████████| 469/469 [01:13<00:00,  6.41it/s]
```

```
Test set: Average loss: 0.0262, Accuracy: 9915/10000 (99.15%)

EPOCH: 2
Loss=0.07649312913417816 Batch_id=468 Accuracy=99.21: 100%|██████████| 469/469 [01:13<00:00,  6.41it/s]

Test set: Average loss: 0.0242, Accuracy: 9915/10000 (99.15%)

EPOCH: 3
Loss=0.001318302471190691 Batch_id=468 Accuracy=99.38: 100%|██████████| 469/469 [01:13<00:00,  6.38it/s]

Test set: Average loss: 0.0309, Accuracy: 9906/10000 (99.06%)

EPOCH: 4
Loss=0.008248790167272091 Batch_id=468 Accuracy=99.51: 100%|██████████| 469/469 [01:13<00:00,  6.38it/s]

Test set: Average loss: 0.0277, Accuracy: 9908/10000 (99.08%)

EPOCH: 5
Loss=0.01677345670759678 Batch_id=468 Accuracy=99.64: 100%|██████████| 469/469 [01:13<00:00,  6.37it/s]

Test set: Average loss: 0.0210, Accuracy: 9930/10000 (99.30%)

EPOCH: 6
Loss=0.01781732775270939 Batch_id=468 Accuracy=99.65: 100%|██████████| 469/469 [01:13<00:00,  6.37it/s]

Test set: Average loss: 0.0211, Accuracy: 9934/10000 (99.34%)

EPOCH: 7
Loss=0.0012531877728179097 Batch_id=468 Accuracy=99.70: 100%|██████████| 469/469 [01:13<00:00,  6.39it/s]

Test set: Average loss: 0.0160, Accuracy: 9949/10000 (99.49%)

EPOCH: 8
Loss=0.0028944441583007574 Batch_id=468 Accuracy=99.74: 100%|██████████| 469/469 [01:13<00:00,  6.38it/s]

Test set: Average loss: 0.0211, Accuracy: 9934/10000 (99.34%)

EPOCH: 9
Loss=0.005427862051874399 Batch_id=468 Accuracy=99.82: 100%|██████████| 469/469 [01:13<00:00,  6.37it/s]
```

The shortcut 'Print notebook' is disabled when a code cell output iframe is active. Use the escape key to leave the iframe and enter the shortcut again.

```
Loss=0.0012176345335319638 Batch_id=468 Accuracy=99.87: 100%|██████████| 469/469 [01:13<00:00,  6.37it/s]

Test set: Average loss: 0.0158, Accuracy: 9952/10000 (99.52%)

EPOCH: 11
Loss=0.0016973119927570224 Batch_id=468 Accuracy=99.88: 100%|██████████| 469/469 [01:13<00:00,  6.38it/s]
```

```
%matplotlib inline
import matplotlib.pyplot as plt

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

```
Text(0.5, 1.0, 'Test Accuracy')
```