Mansoor Nabawi, 309498

References:

- https://www.askpython.com/python-modules/pytorch-custom-datasets
- https://debuggercafe.com/custom-dataset-and-dataloader-in-pytorch/

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data

import torchvision.transforms as transforms
import torchvision.datasets as datasets

from sklearn import decomposition
from sklearn import manifold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from tqdm.notebook import tqdm, trange
import matplotlib.pyplot as plt
import numpy as np

import copy
import random
import time

from torch.utils.tensorboard import SummaryWriter
%load_ext tensorboard
```

```
SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

loading and normalizing data

```
ROOT = '.data'

train_data = datasets.MNIST(root=ROOT,
                            train=True,
                            download=True)

mean = train_data.data.float().mean() / 255
std = train_data.data.float().std() / 255
```

```
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.gz
  0%|          | 0/9912422 [00:00<?, ?it/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.gz
  0%|          | 0/28881 [00:00<?, ?it/s]
Extracting .data/MNIST/raw/train-labels-idx1-ubyte.gz to .data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to .data/MNIST/raw/t10k-images-idx3-ubyte.gz
  0%|          | 0/1648877 [00:00<?, ?it/s]
Extracting .data/MNIST/raw/t10k-images-idx3-ubyte.gz to .data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to .data/MNIST/raw/t10k-labels-idx1-ubyte.gz
  0%|          | 0/4542 [00:00<?, ?it/s]
Extracting .data/MNIST/raw/t10k-labels-idx1-ubyte.gz to .data/MNIST/raw
```

defining some transformations for test and train

```
train_transforms = transforms.Compose([
                            transforms.RandomRotation(5, fill=(0,)),
                            transforms.RandomCrop(28),
                            transforms.ToTensor(),
                            transforms.Normalize(mean=[mean], std=[std])
                                  ])

test_transforms = transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize(mean=[mean], std=[std])
                                     ])
```

```
train_data = datasets.MNIST(root=ROOT,
                            train=True,
                            download=True,
                            transform=train_transforms)

test_data = datasets.MNIST(root=ROOT,
                           train=False,
                           download=True,
                           transform=test_transforms)
```

writing the custom dataset

- I am going to get the dataset in my custom class.
- we make random indexes of data but in a smaller size.
- we choose the random indexes of data and put it in another variables.
- Then i will concat/stack based on the number of k we have.

- 1later this class will be used by train iterator.

```
import numpy as np
from torch.utils.data import Dataset

class custom_data(Dataset):
    def __init__(self, data, k):
        self.data = data
        self.lendata = len(self.data)
        self.k = k
        self.get_index()


    def __len__(self):
        return len(self.data)//self.k

    def __getitem__(self, idx):
      indx = self.idx_[idx]
      x = []
      y = []
      for i in indx:
        x.append(self.data.__getitem__(i)[0])
        y.append(self.data.__getitem__(i)[1])

      data_, label = torch.stack(x), np.sum(y).astype(dtype=np.float32)
      return data_, label

    def get_index(self):
        self.idx_ = np.random.randint(0,self.lendata, size=(self.lendata//self.k, self.k))
```

```
np.random.randint(0,len(train_data), size=(len(train_data)//3, 3)).shape
```

```
(20000, 3)
```

```
train_cdata = custom_data(train_data,k = 3)
test_cdata = custom_data(test_data,k = 3)
```

```
len(train_cdata)
```

```
20000
```

```
BATCH_SIZE = 64

train_iterator = data.DataLoader(train_cdata,
                                 shuffle=True,
                                 num_workers=2,
                                 batch_size=BATCH_SIZE)

test_iterator = data.DataLoader(test_cdata,
                                num_workers=2,
                                batch_size=BATCH_SIZE,
                               shuffle = False)
```

```
example = enumerate(train_iterator)
idx, (dt, lb) = next(example)
```

N,K,C,W,H

```
dt.shape
```

```
torch.Size([64, 3, 1, 28, 28])
```

```
lb.shape
```

```
torch.Size([64])
```

we make few changes to our NN.

we add another layer for the final prediction, as we have ten numbers but at the end we will have sum of the three numbers and it is going to be only one output.

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

        self.conv1 = nn.Conv2d(in_channels=1,
                               out_channels=6,
                               kernel_size=5,
                               padding=2)

        self.conv2 = nn.Conv2d(in_channels=6,
                               out_channels=16,
                               kernel_size=5)

        self.fc_1 = nn.Linear(16 * 5 * 5, 120)
        self.fc_2 = nn.Linear(120, 84)
        self.fc_3 = nn.Linear(84, 10)
        self.fc_4 = nn.Linear(10,1)

    def forward(self, x):
        N,K,C,W,H = x.size()
        x = x.view(N*K,C,W,H)

        x = self.conv1(x)
        x = F.max_pool2d(F.relu(x), kernel_size=2)


        x = self.conv2(x)
        x = F.relu(x)

        x = F.max_pool2d(x, kernel_size=2)
        x = x.view(-1,16*5*5 )

        x = self.fc_1(x)
        x = F.relu(x)
```

```
        x = self.fc_2(x)
        x = F.relu(x)

        x = self.fc_3(x)
        x = F.relu(x)

        x = self.fc_4(x)
        x = F.relu(x)
        x = x.view(N,K)
        x = torch.sum(x, dim=1)


        return x
```

```
print(LeNet())
```

```
LeNet(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc_1): Linear(in_features=400, out_features=120, bias=True)
  (fc_2): Linear(in_features=120, out_features=84, bias=True)
  (fc_3): Linear(in_features=84, out_features=10, bias=True)
  (fc_4): Linear(in_features=10, out_features=1, bias=True)
)
```

```
def train(model, iterator, optimizer, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for  (x, y) in tqdm(iterator, desc="Training", leave=False):

        x = x.to(device)
        y = y.to(device)

        optimizer.zero_grad()

        y_pred = model(x)

        loss = criterion(y_pred, y)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()


    return epoch_loss / len(iterator)
```

```
def test(model, iterator, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for  (x, y) in tqdm(iterator, desc="Testing", leave=False):

          x = x.to(device)
          y = y.to(device)

          y_pred = model(x)

          loss = criterion(y_pred, y)

          epoch_loss += loss.item()


    return epoch_loss / len(iterator)
```

```
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

```
logs = 'runs/mnist_custom'
tb = SummaryWriter(logs)
```

```
selected_optimizer='Adam'
for learning_rate in [0.001]:


    model = LeNet()

    optimizer = optim.Adam(model.parameters())

    criterion = nn.MSELoss()

    #Checking if we can use GPU

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    model = model.to(device)
    criterion = criterion.to(device)




    avail_optimizers = {'Adam':torch.optim.Adam(model.parameters(), lr=learning_rate, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False),
                        'RMS': torch.optim.RMSprop(model.parameters(), lr=learning_rate, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0, centered=False),
                        'SGD': torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0, dampening=0, weight_decay=0, nesterov=False)}

    optimizer = avail_optimizers[selected_optimizer]

    epochs = 20
```

```
print(f"selected optimizer and learning rate: {optimizer}")
for n_iter in tqdm(range(epochs)):
    print(f"Epoch {n_iter+1}/{epochs}")


    start_time = time.monotonic()

    train_loss = train(model, train_iterator, optimizer, criterion, device)
    train_iterator.dataset.get_index()
    test_loss = test(model, test_iterator, criterion, device)
    test_iterator.dataset.get_index()


    end_time = time.monotonic()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    print(f'Epoch: {n_iter+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} ')
    print(f'\t Test. Loss: {test_loss:.3f} ')

    if tb is not None:
        tb.add_scalars(f'Loss_{learning_rate}', {"Train":train_loss,
                                "Test":test_loss}, n_iter)

        tb.add_scalars(f'Train_loss', {f"{learning_rate}":train_loss}, n_iter)
        tb.add_scalars(f'Test_loss', {f"{learning_rate}":test_loss}, n_iter)
```

```
selected optimizer and learning rate: Adam (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    eps: 1e-08
    lr: 0.001
    maximize: False
    weight_decay: 0
)
100%                                    20/20 [11:00<00:00, 33.62s/it]
Epoch 1/20
Epoch: 01 | Epoch Time: 0m 32s
        Train Loss: 24.447
         Test. Loss: 9.513
Epoch 2/20
Epoch: 02 | Epoch Time: 0m 33s
        Train Loss: 7.838
         Test. Loss: 6.049
Epoch 3/20
Epoch: 03 | Epoch Time: 0m 32s
        Train Loss: 4.807
         Test. Loss: 3.873
Epoch 4/20
Epoch: 04 | Epoch Time: 0m 33s
        Train Loss: 3.493
         Test. Loss: 3.012
Epoch 5/20
Epoch: 05 | Epoch Time: 0m 32s
        Train Loss: 2.794
         Test. Loss: 3.110
Epoch 6/20
Epoch: 06 | Epoch Time: 0m 32s
        Train Loss: 2.332
         Test. Loss: 2.150
Epoch 7/20
Epoch: 07 | Epoch Time: 0m 33s
        Train Loss: 2.001
         Test. Loss: 2.116
Epoch 8/20
Epoch: 08 | Epoch Time: 0m 32s
        Train Loss: 1.804
         Test. Loss: 1.992
Epoch 9/20
Epoch: 09 | Epoch Time: 0m 32s
        Train Loss: 1.587
         Test. Loss: 1.536
Epoch 10/20
Epoch: 10 | Epoch Time: 0m 32s
        Train Loss: 1.500
         Test. Loss: 1.480
Epoch 11/20
Epoch: 11 | Epoch Time: 0m 32s
        Train Loss: 1.349
         Test. Loss: 1.436
Epoch 12/20
Epoch: 12 | Epoch Time: 0m 33s
        Train Loss: 1.249
         Test. Loss: 1.299
Epoch 13/20
Epoch: 13 | Epoch Time: 0m 32s
        Train Loss: 1.134
         Test. Loss: 1.298
Epoch 14/20
Epoch: 14 | Epoch Time: 0m 32s
        Train Loss: 1.042
         Test. Loss: 1.573
Epoch 15/20
Epoch: 15 | Epoch Time: 0m 32s
        Train Loss: 1.110
         Test. Loss: 1.180
Epoch 16/20
Epoch: 16 | Epoch Time: 0m 33s
        Train Loss: 1.028
         Test. Loss: 1.199
Epoch 17/20
Epoch: 17 | Epoch Time: 0m 33s
        Train Loss: 0.867
         Test. Loss: 1.031
Epoch 18/20
Epoch: 18 | Epoch Time: 0m 32s
        Train Loss: 0.813
         Test. Loss: 1.159
Epoch 19/20
Epoch: 19 | Epoch Time: 0m 35s
        Train Loss: 0.857
         Test. Loss: 1.075
Epoch 20/20
Epoch: 20 | Epoch Time: 0m 33s
        Train Loss: 0.759
         Test. Loss: 1.519
```

As it is visible our model is performing very well and the loss is decreasing

```
%tensorboard --logdir runs/mnist_custom
```