In [1]:

```python
try:
    import torch as t
    import torch.nn as tnn
except ImportError:
    print("Colab users: pytorch comes preinstalled. Select Change Ru")
    print("Local users: Please install pytorch for your hardware using instr
    print("ACG users: Please follow instructions here: https://vikasdhiman.i

    raise
```

In [2]:

```python
def wget(url, filename):
    """
    Download files using requests package.
    Better than wget command line because this is cross platform.
    """
    try:
        import requests
    except ImportError:
        import subprocess
        subprocess.call("pip install requests".split())
        import requests
    r = requests.get(url)
    with open(filename, 'wb') as fd:
        for chunk in r.iter_content():
            fd.write(chunk)
```

In [3]:

```python
## Doing it the Pytorch way without using our custom feature extraction
DEVICE='cuda:0'
DTYPE=t.float32
import torch
import torch.nn
import torch.optim
import torchvision
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader

torch.manual_seed(17)

# Getting the dataset, the Pytorch way
all_training_data = torchvision.datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = torchvision.datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

In [4]:

```
training_data, validation_data = torch.utils.data.random_split(all_training_
```

In [8]:

```
# Hyper parameters
learning_rate = 1e-3 # controls how fast the
batch_size = 64
epochs = 5
momentum = 0.9

training_dataloader = DataLoader(training_data, shuffle=True, batch_size=bat
validation_dataloader = DataLoader(validation_data,  batch_size=batch_size)
test_dataloader = DataLoader(test_data,  batch_size=batch_size)


loss = torch.nn.CrossEntropyLoss()

# TODO:
# Define model = ?
class MLPNetwork(torch.nn.Module):
    def __init__(self, hidden_size=10, nclasses=10, input_size=28*28):
        super().__init__()
        self._layers = torch.nn.ModuleList([torch.nn.Flatten(),
            tnn.Linear(input_size, hidden_size),
            tnn.ReLU(),
            tnn.Linear(hidden_size, nclasses)])
    def forward(self, x):
        for l in self._layers:
            xnext = l(x) # call the layers in sequence
            x = xnext
        return x
model = MLPNetwork()

# alternatively you can also
# hidden_size=10
# nclasses=10
# input_size=28*28
# model = torch.nn.Sequential(torch.nn.Flatten(),
#             tnn.Linear(input_size, hidden_size),
#             tnn.ReLU(),
#             tnn.Linear(hidden_size, nclasses))
#

# Define optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=m

def loss_and_accuracy(model, loss, validation_dataloader, device=DEVICE):
    # Validation loop
    validation_size = len(validation_dataloader.dataset)
    num_batches = len(validation_dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in validation_dataloader:
            X = X.to(device)
            y = y.to(device)
            pred = model(X)
            test_loss += loss(pred, y).item()
            correct += (pred.argmax(dim=-1) == y).type(DTYPE).sum().item()

    test_loss /= num_batches
    correct /= validation_size
    return test_loss, correct

def train(model, loss, training_dataloader, validation_dataloader, device=DE
    model.to(device)
    train_losses = []
    valid_losses = []
    for t in range(epochs):
        # Train loop
        training_size = len(training_dataloader.dataset)
        for batch, (X, y) in enumerate(training_dataloader):
            X = X.to(device)
            y = y.to(device)
            # Compute prediction and loss
            pred = model(X)
            loss_t = loss(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss_t.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_t, current = loss_t.item(), (batch + 1) * len(X)
                print(f"loss: {loss_t:>7f}  [{current:>5d}/{training_size:>5
                train_losses.append(loss_t)
                valid_loss, correct = loss_and_accuracy(model, loss, validat
                valid_losses.append(valid_loss)
                print(f"Validation Error: \n Accuracy: {(100*correct):>0.1f}
    return model, train_losses, valid_losses

trained_model, train_losses, valid_losses = train(model, loss, training_data

test_loss, correct = loss_and_accuracy(model, loss, test_dataloader)
print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_los
Validation Error:   64/54000]
 Accuracy: 10.9%, Avg loss: 2.325170

Validation Error: 6464/54000]
 Accuracy: 13.6%, Avg loss: 2.183869

Validation Error: 2864/54000]
 Accuracy: 34.1%, Avg loss: 2.001862

Validation Error: 9264/54000]
 Accuracy: 46.8%, Avg loss: 1.787784

Validation Error: 5664/54000]
 Accuracy: 54.8%, Avg loss: 1.580385

Validation Error: 2064/54000]
 Accuracy: 62.2%, Avg loss: 1.389772

Validation Error: 8464/54000]
 Accuracy: 68.5%, Avg loss: 1.216087

Validation Error: 4864/54000]
 Accuracy: 75.8%, Avg loss: 1.061541

Validation Error: 1264/54000]
 Accuracy: 79.9%, Avg loss: 0.935546

Validation Error:   64/54000]
 Accuracy: 80.2%, Avg loss: 0.889401

Validation Error: 6464/54000]
 Accuracy: 82.1%, Avg loss: 0.803695

Validation Error: 2864/54000]
 Accuracy: 82.9%, Avg loss: 0.736784

Validation Error: 9264/54000]
 Accuracy: 83.7%, Avg loss: 0.682686

Validation Error: 5664/54000]
 Accuracy: 84.5%, Avg loss: 0.639954

Validation Error: 2064/54000]
 Accuracy: 85.1%, Avg loss: 0.606284

Validation Error: 8464/54000]
 Accuracy: 85.5%, Avg loss: 0.577947

Validation Error: 4864/54000]
 Accuracy: 86.0%, Avg loss: 0.553612

Validation Error: 1264/54000]
 Accuracy: 86.4%, Avg loss: 0.534344

Validation Error:   64/54000]
 Accuracy: 86.7%, Avg loss: 0.527205
```

```
Validation Error: 6464/54000]
 Accuracy: 86.8%, Avg loss: 0.510988

Validation Error: 2864/54000]
 Accuracy: 87.1%, Avg loss: 0.497435

Validation Error: 9264/54000]
 Accuracy: 87.0%, Avg loss: 0.485881

Validation Error: 5664/54000]
 Accuracy: 87.4%, Avg loss: 0.474297

Validation Error: 2064/54000]
 Accuracy: 87.4%, Avg loss: 0.465911

Validation Error: 8464/54000]
 Accuracy: 87.7%, Avg loss: 0.456387

Validation Error: 4864/54000]
 Accuracy: 87.9%, Avg loss: 0.449242

Validation Error: 1264/54000]
 Accuracy: 87.9%, Avg loss: 0.442204

Validation Error:   64/54000]
 Accuracy: 88.2%, Avg loss: 0.439258

Validation Error: 6464/54000]
 Accuracy: 88.1%, Avg loss: 0.433261

Validation Error: 2864/54000]
 Accuracy: 88.0%, Avg loss: 0.428486

Validation Error: 9264/54000]
 Accuracy: 88.3%, Avg loss: 0.423615

Validation Error: 5664/54000]
 Accuracy: 88.5%, Avg loss: 0.420128

Validation Error: 2064/54000]
 Accuracy: 88.5%, Avg loss: 0.414353

Validation Error: 8464/54000]
 Accuracy: 88.4%, Avg loss: 0.409636

Validation Error: 4864/54000]
 Accuracy: 88.7%, Avg loss: 0.407865

Validation Error: 1264/54000]
 Accuracy: 88.8%, Avg loss: 0.401992

Validation Error:   64/54000]
 Accuracy: 88.9%, Avg loss: 0.401415

Validation Error: 6464/54000]
```

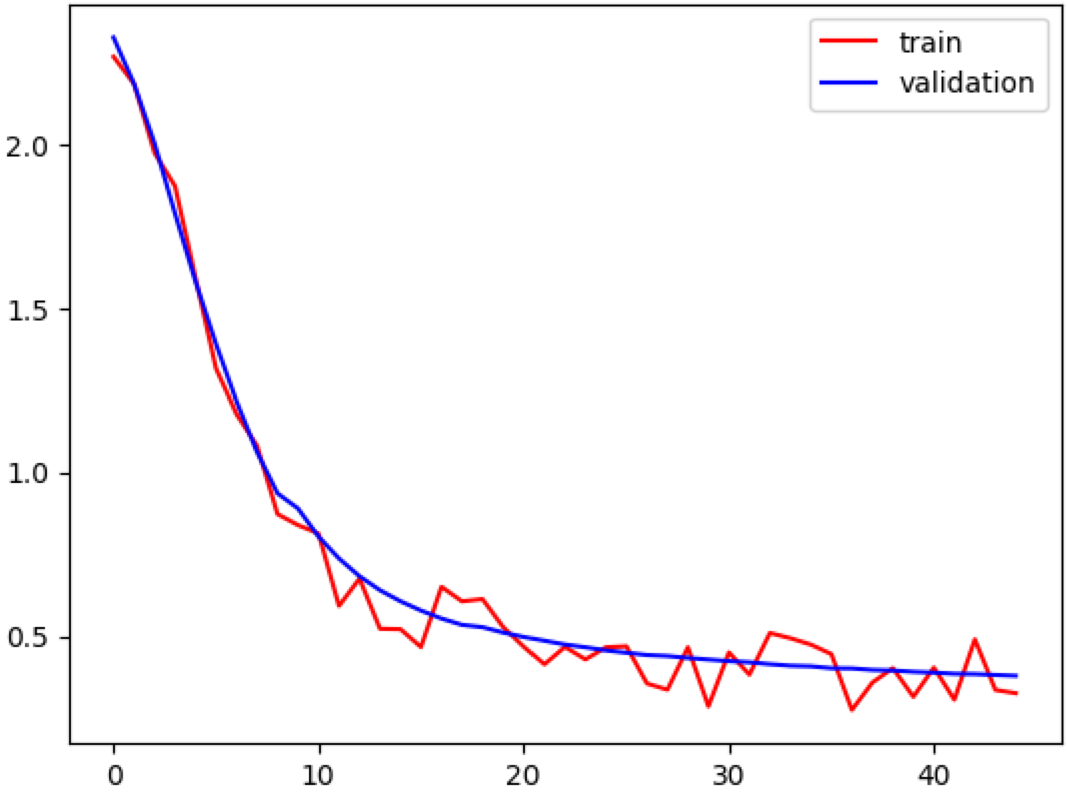
```
 Accuracy: 88.9%, Avg loss: 0.396768

Validation Error: 2864/54000]
 Accuracy: 88.9%, Avg loss: 0.395058

Validation Error: 9264/54000]
 Accuracy: 89.0%, Avg loss: 0.391086

Validation Error: 5664/54000]
 Accuracy: 88.9%, Avg loss: 0.388569

Validation Error: 2064/54000]
 Accuracy: 89.0%, Avg loss: 0.385012

Validation Error: 8464/54000]
 Accuracy: 89.0%, Avg loss: 0.384191

Validation Error: 4864/54000]
 Accuracy: 89.3%, Avg loss: 0.381349

Validation Error: 1264/54000]
 Accuracy: 89.3%, Avg loss: 0.378932

Test Error:
 Accuracy: 90.0%, Avg loss: 0.351287
```

In [10]:

```python
import matplotlib.pyplot as plt
plt.plot(train_losses, 'r', label='train')
plt.plot(valid_losses, 'b', label='validation')
plt.legend()
```

Out[10]:

```
<matplotlib.legend.Legend at 0x7f92d52b5900>
```



```
In [11]: torch.nn.__file__
```

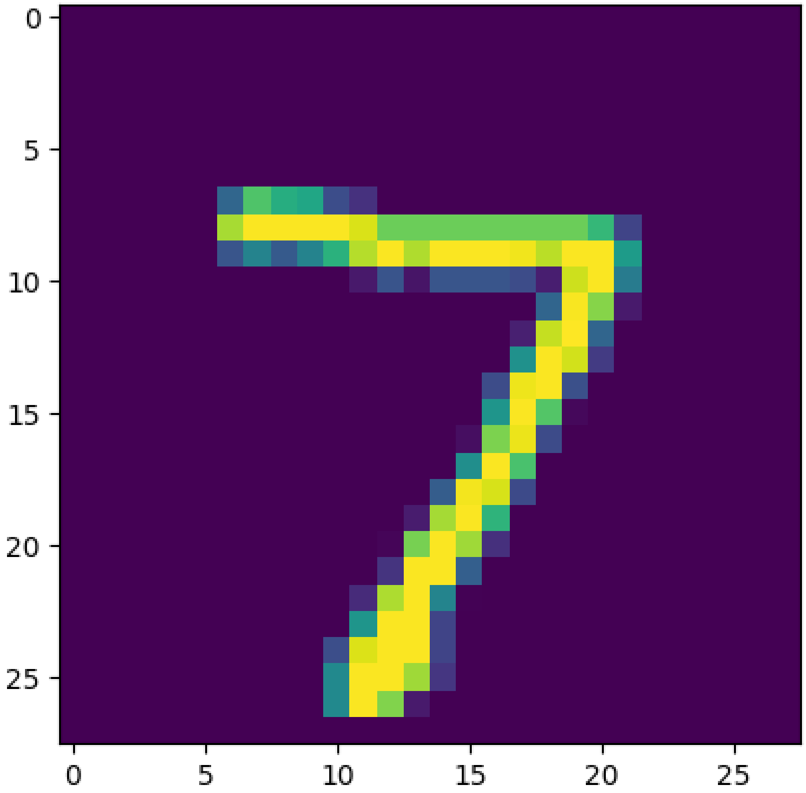
```
Out[11]: '/home/vdhiman/.local/share/virtualenvs/nbgrader-notebooks-_16a_jDm/lib/pyt
         hon3.10/site-packages/torch/nn/__init__.py'
```

```
In [12]: X, _ = next(iter(test_dataloader))
         X.shape
```

```
Out[12]: torch.Size([64, 1, 28, 28])
```

```
In [13]: import matplotlib.pyplot as plt
         plt.imshow(X[0, 0])
```

```
Out[13]: <matplotlib.image.AxesImage at 0x7f92d02888b0>
```



```
In [14]: print("The predicted image label is ", model(X.to(DEVICE)).argmax(dim=-1)[0]
```

```
The predicted image label is  7
```

```
In [ ]:
```