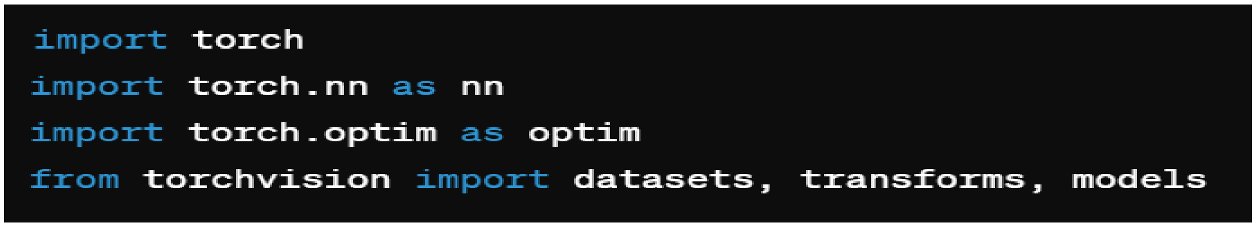
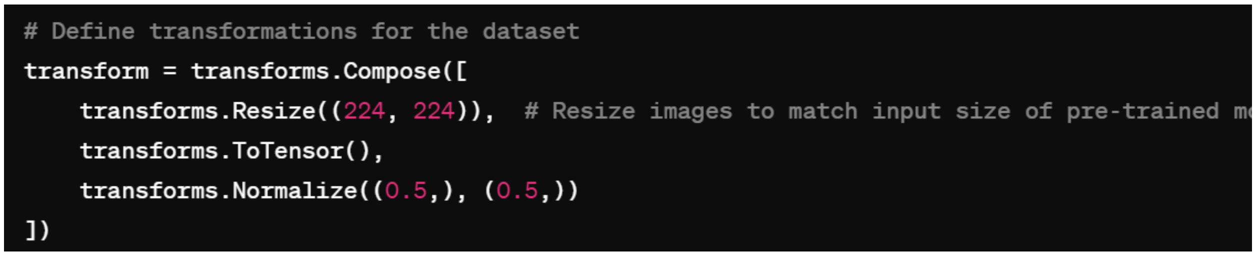
Assignment #10

For all your work, submit a Notebook (either Jupyter or Colab.)

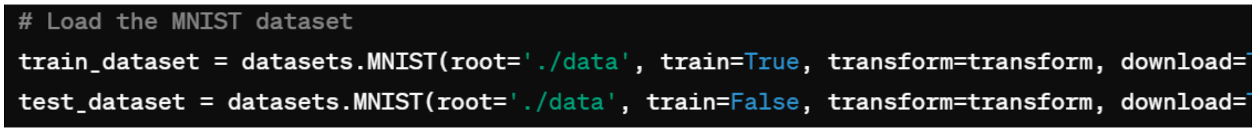
Utilizing Python code snippets demonstrate how to use transfer learning with a **pre-trained (Resnet *) model in PyTorch framework**, using the MNIST dataset as an example:



```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
```

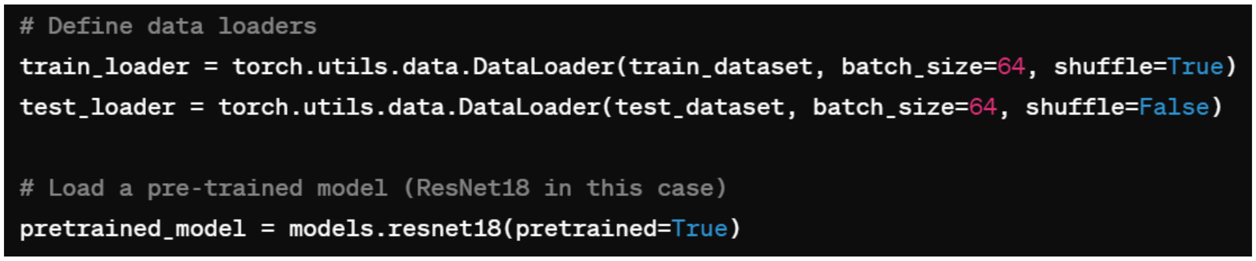


```
# Define transformations for the dataset
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to match input size of pre-trained m
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```



```
# Load the MNIST dataset
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=
```

Note: When downloading and loading the training set with torchvision.datasets.MNIST, specifying the root directory where the data will be saved (e.g., ‘data’, whether it's the training set (train=True), and the defined transformations.



```
# Define data loaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False)

# Load a pre-trained model (ResNet18 in this case)
pretrained_model = models.resnet18(pretrained=True)
```

Perform the following tasks:

- First define **transformations** for the MNIST dataset,
- **Load** the dataset using torchvision
- Define **data loaders** for training and testing.
- Load a **pre-trained ResNet18 model** and freeze its convolutional layers.
- **Modify the fully connected layer** of the pre-trained model to output 10 classes (corresponding to the classes in MNIST).
- Define the **loss function** (cross-entropy) and **optimizer** (Adam).
- **Train** the model for a specified number of epochs (10)
- Compute the **loss** and **updating the weights** using **backpropagation**.
- **Evaluate** the trained model on the test set and compute its accuracy.

(*) **ResNet**, short for **Residual Network**, is a deep convolutional neural network architecture proposed by researchers at Microsoft Research. It addresses the problem of vanishing gradients encountered in training very deep neural networks by introducing skip connections or "shortcut connections" that allow gradients to flow more directly through the network during training.

The key innovation in ResNet is the introduction of residual blocks. These blocks contain identity mappings, where the input to a block is added to the output of the block. This concept of residual learning helps to mitigate the degradation problem that occurs when adding more layers to a neural network doesn't necessarily lead to improved performance. By introducing these shortcut connections, ResNet enables the training of very deep networks (e.g., hundreds of layers) without suffering from the vanishing gradient problem.

ResNet architectures come in various depths, such as ResNet-50, ResNet-101, and ResNet-152, which have 50, 101, and 152 layers respectively. These architectures have been pre-trained on large datasets like ImageNet for tasks such as image classification, object detection, and image segmentation. ResNet models have demonstrated state-of-the-art performance on various computer vision tasks and have become widely adopted in both research and industry due to their effectiveness and ease of training.