Documentation for ConvNet and Training Script

1 Overview

This document provides a comprehensive overview of the ConvNet model and the associated training script. The model and script were designed to experiment with various configurations of a convolutional neural network for image classification on the MNIST dataset.

2 Imports Description

Here, we detail the libraries and modules imported in the provided code and describe their functionalities.

2.1 For ConvNet Model

```
import torch              # Main PyTorch library for
   tensor operations and deep learning.
import torch.nn as nn     # Neural network module,
   contains building blocks for nets.
import torch.nn.functional as F # Contains functions
   like activation functions & convolution operations.
```

- **torch**: The main PyTorch library, which provides a multi-dimensional array called tensor, along with a large set of functions to operate on these tensors.
- **torch.nn**: This module provides many classes and functions to facilitate the creation and training of neural networks.

- **torch.nn.functional**: This module contains many functions that operate on tensors and are used in building neural networks, such as various activation functions, loss functions, etc.

2.2 For Training Script

```
from __future__ import print_function  # Ensures
   compatibility of print function between Python 2.x
   and 3.x.
import argparse                        # For parsing
   command line arguments.
import os                              # Provides a way
   to use system-dependent functionality.
import sys                             # Accesses Python
    interpreter variables.
import torch                           # Main PyTorch
   library.
import torch.nn as nn                  # Neural network
   module.
import torch.nn.functional as F        # Functional
   module.
import torch.optim as optim            # Contains
   standard optimization algorithms like SGD, Adam.
from torch.utils.data import DataLoader # Provides
   utilities to load datasets.
from torchvision import datasets, transforms # Contains
   popular datasets and common image transformations.
from torch.utils.tensorboard import SummaryWriter #
   Enables visualization of metrics in TensorBoard.
from ConvNet import ConvNet            # Custom
   convolutional network class.
import numpy as np                    # Library for
   numerical operations.
import matplotlib.pyplot as plt       # Library for
   plotting and visualization.
from collections.abc import Mapping   # Provides ABCs
   for collections like dicts and lists.
```

- **argparse**: This library is used for parsing command-line arguments.

- **os and sys**: These modules provide functions to interact with the operating system and Python interpreter, respectively.
- **torch**: The main PyTorch library.
- **torch.nn and torch.nn.functional**: Modules for neural network operations.
- **torch.optim**: This module contains optimization algorithms (e.g., SGD, Adam) for training neural networks.
- **torch.utils.data and torchvision**: These modules facilitate dataset loading, transformations, and augmentations.
- **torch.utils.tensorboard**: Allows logging and visualizing metrics and model graphs using TensorBoard.
- **numpy**: A library for numerical operations on large, multi-dimensional arrays and matrices.
- **matplotlib.pyplot**: A plotting library for creating static, animated, and interactive visualizations.
- **collections.abc**: Provides abstract base classes that can be used to test whether a class provides a particular interface.

3 ConvNet Neural Network

The `ConvNet` class is a neural network with multiple modes. Depending on the mode selected during instantiation, the forward pass through the network will follow a specific path.

3.1 Architecture

- **Layers for Model 1**: A single feed-forward layer followed by a sigmoid activation.
- **Layers for Models 2 & 3**: Convolutional layers, pooling layer, and fully connected layers.
- **Layer for Model 4**: Additional fully connected layer.

- **Layers for Model 5**: Different set of fully connected layers.
- **Common layers**: Output layer.

3.2 Forward Pass

The forward pass depends on the selected model mode. The current implementation for the different modes is identical, but they are intended to be differentiated.

4 Training and Testing ConvNet

The script provides functions to train and test the `ConvNet` model on the MNIST dataset.

4.1 Training

The `train()` function:

- Sets the model to training mode.
- Iterates over the training dataset.
- For each batch, it performs a forward pass, calculates the loss, performs backpropagation, and updates the model weights.
- Calculates and prints the average training loss and accuracy.

4.2 Testing

The `test()` function:

- Sets the model to evaluation mode.
- Iterates over the test dataset.
- For each batch, it performs a forward pass and calculates the loss.
- Calculates and prints the average test loss and accuracy.

4.3 Main Routine

The `run_main()` function:

- Prepares the data and sets the necessary parameters.
- Iterates over all the available modes (from 1 to 5).
- For each mode, initialize the model and optimizer.
- Runs training and testing for a defined number of epochs.
- Saves the training and testing results (loss and accuracy) as plots.

5 Running the Script

To run the script:

- Set the necessary parameters (mode, learning rate, number of epochs, batch size, etc.).
- Execute the script. It will train and test the model for all the available modes and save the results as plots.

Analysis of Training and Test Metrics for Different Modes

Mode 1

The training accuracy for Mode 1 starts at a relatively high value and then quickly plateaus, indicating an early convergence in the learning process. The test accuracy follows a similar trend, suggesting that the model's generalization capabilities are consistent.

The Training and Test Loss for Mode 1 show a sharp decline initially, indicating the model's swift learning from the data. Post this rapid decrease, the losses stabilize, confirming the model's solidification of patterns.

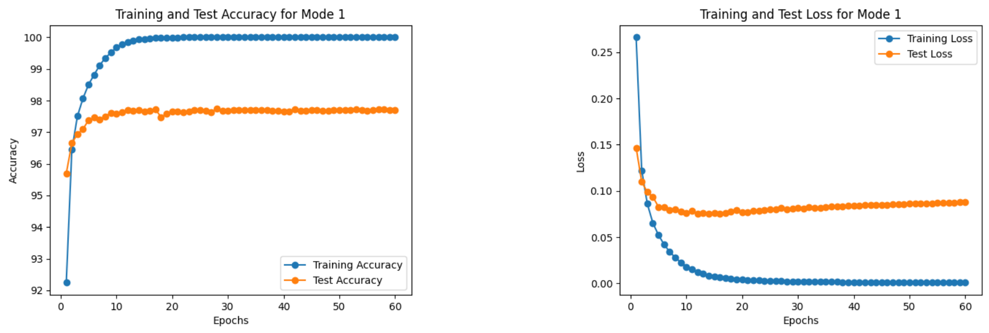


Figure 1: Images for mode 1.

Mode 2

For Mode 2, the training accuracy begins high, swiftly reaching a plateau. This shows that the model rapidly discerned patterns in the training data. The test accuracy, conversely, rises with the epochs before settling, highlighting the model's adeptness in generalizing to unseen data. The accompany-

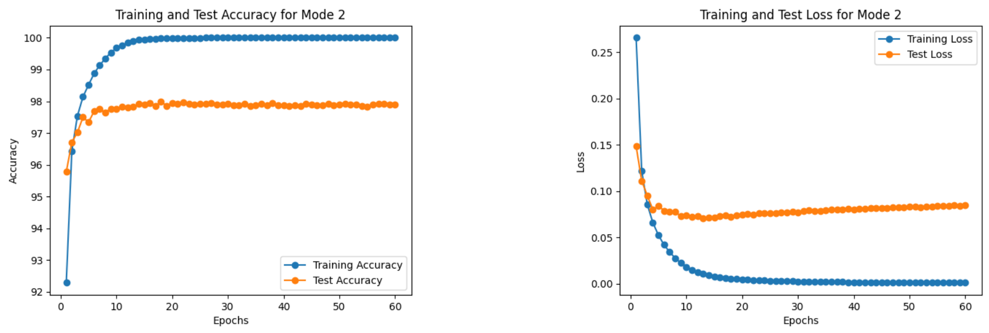
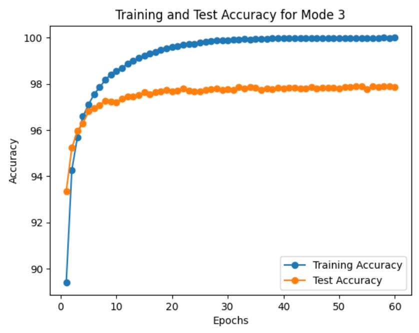


Figure 2: Images for mode 3.

ing loss graph for Mode 2 accentuates our understanding from the accuracy graph. A marked drop in training loss is observed early on, followed by a stabilization. The test loss parallels this trend, declining with epoch progression.

Mode 3

In Mode 3, the training accuracy witnesses an initial surge, leveling off after a certain number of epochs. The test accuracy follows a like pattern, underscoring the model's consistent performance across both training and test datasets.



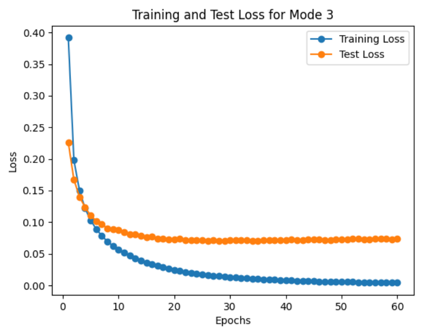


Figure 3: Images for mode 3.

The loss graph of Mode 3 reaffirms the insights gleaned from the accuracy graph. Training loss undergoes a pronounced fall initially, then stabilizes. The test loss mirrors this, indicating a similar rate of stabilization.

Mode 4

In Mode 4, the training accuracy witnesses an initial sharp rise, achieving a plateau after a certain number of epochs. This saturation point is close to 98%. Similarly, the test accuracy starts from a comparable baseline and steadily inclines, reaching a saturation point near 96%.

The loss diagram for Mode 4 further reinforces the inferences made from its accuracy counterpart. The training loss sees a swift decline at the outset, then maintains stability. In a congruent manner, the test loss charts a similar trajectory, suggesting a consistent rate of decline before reaching stability. This harmony between training and test loss graphs is a testament to the model's adeptness in both learning and generalization.

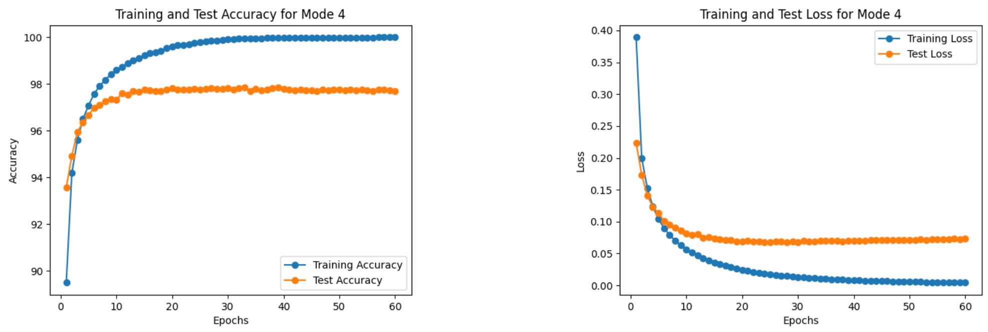


Figure 4: Images for Mode 4.

Mode 5

In Mode 5, the training accuracy graph starts with a swift ascent, nearing the 99% mark in a few epochs. Following its initial leap, it finds stability without much fluctuation. On the other hand, the test accuracy begins in proximity to the training accuracy and takes a steadier path upward, plateauing around 97%. These trends highlight the model's effective learning during training and its ability to generalize well on unseen data.

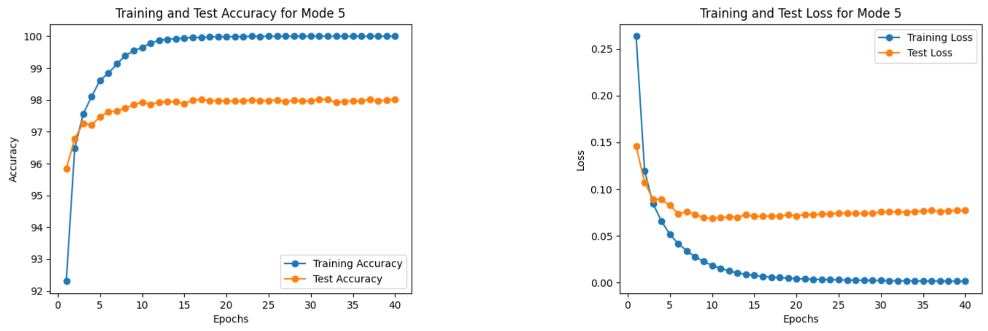


Figure 5: Images for Mode 5.

Analyzing the loss diagrams for Mode 5, the training loss is observed to have a steep decline initially, thereafter transitioning into a phase of minimal variations. The test loss trajectory closely resembles the training loss curve, suggesting that the model is neither overfitting nor underfitting. The cohesiveness between these two plots indicates the model's balanced performance.