Harshitha J
B00979447

MNIST Image Classification

Introduction & Proposal:

This code shows how to create and train neural network models for image classification using the MNIST dataset using PyTorch and TensorFlow. The objective is to assess how well PyTorch-based models (Convolutional Neural Network and Feedforward Neural Network) perform in comparison to a Keras-based TensorFlow model.

Methodology:

- **Data Preprocessing:**
  - o The code starts by defining data transformations to use PyTorch's transforms to normalise the pixel values of the MNIST pictures.Compose.
  - o The MNIST dataset is loaded, divided into training and testing sets, and data loaders are made using PyTorch's torch for batch processing.utils.data.DataLoader.
- **TensorFlow Model:**
  - o The Keras API of TensorFlow is used to define a basic neural network model. This model has a flattening layer, a dense hidden layer activated by ReLU, and an output layer activated by softmax.
  - o A sparse categorical cross-entropy loss and an Adam optimizer are used to build the TensorFlow model.
- **PyTorch CNN Model:**
  - o CNNNet, a Convolutional Neural Network (CNN) model, is created with PyTorch. Convolutional, max-pooling, and fully linked layers make up the CNN architecture.
  - o A CNN model instance is constructed together with an Adam optimizer and a CrossEntropyLoss loss function.
- **PyTorch FNN Model:**
  - o PyTorch is also used to create a Feedforward Neural Network (FNN) model referred to as FNNNet. Three completely linked layers make to this FNN's design.
  - o A CrossEntropyLoss loss function and Adam, an optimizer, are constructed together with an instance of the FNN model.
  - o • Looping Training:
  - o Separate training loops are implemented for the TensorFlow, PyTorch CNN, and PyTorch FNN models.

  - For each model, it iterates through the training data, calculates the loss, backpropagates gradients, and updates model weights using the respective optimizer.
  - The training loops are executed for a specified number of epochs (5 in this case).
- **Testing and Accuracy Calculation:**
  - After training all models, they are tested on the MNIST test dataset.
  - The code calculates and reports the accuracy of each model on the test images.

Evaluation & Results:

- The TensorFlow model achieved a test accuracy of approximately 96.6% on the MNIST dataset after training for 5 epochs.
- The PyTorch CNN model achieved a test accuracy of approximately 98.5%.
- The PyTorch FNN model achieved a test accuracy of approximately 96.5%.
- **Discussion:**
- The PyTorch CNN model outperformed both the TensorFlow model and the PyTorch FNN model. This is expected, as CNNs are well-suited for image classification tasks, capturing spatial features.
- The TensorFlow model performed reasonably well, demonstrating that Keras provides an accessible interface for building neural networks.
- The code serves as an example of how to implement and train neural network models using both PyTorch and TensorFlow/Keras and highlights the performance differences among these models.

Conclusion:

In this code, we successfully implemented and compared three neural network models for MNIST image classification using PyTorch and TensorFlow/Keras. While the PyTorch CNN model achieved the highest accuracy, the TensorFlow/Keras model also performed well. This demonstrates the flexibility and power of deep learning frameworks for developing machine learning models. The code can be a valuable reference for those interested in exploring both PyTorch and TensorFlow for image classification tasks.