Documentation

In this documentation will contain 2 parts:

1. Model for MNIST Challenge
2. Experiment Management System

Model for MNIST Challenge

Overview

This script defines and trains a Convolutional Neural Network (CNN) model for the MNIST dataset, which consists of handwritten digits (0-9). The model is built using TensorFlow and Keras, and is designed to classify images into one of the ten digit classes.

Dependencies (Only for the model)

- TensorFlow
- Keras (included within TensorFlow)

Dataset

- **MNIST Dataset**: A collection of 28x28 pixel grayscale images of handwritten digits (0-9). The dataset is divided into a training set and a testing set.

Preprocessing

- **Normalization**: Pixel values of the images are normalized to be in the range of 0 to 1 to aid in the training process.

Model Architecture

1. **Input Layer**: Accepts images of shape (28, 28, 1).
2. **Convolutional Layers**:

    - First Layer: Convolutional layer with `num_filters` (default: 32) filters of size `kernel_size` (default: (3, 3)) followed by a (2, 2) max pooling layer.
    - Second Layer: Convolutional layer with `num_filters * 2` filters followed by a (2, 2) max pooling layer.
3. **Flattening Layer**: Flattens the output from the convolutional layers.
4. **Fully Connected Layers**:
    - Dense layer with `dense_units` (default: 128) neurons and ReLU activation.
    - Dropout layer with 0.5 dropout rate to reduce overfitting.
    - Output Dense layer with 10 neurons (one for each digit) and softmax activation.
5. **Output Layer**: Produces a probability distribution over the 10 digit classes.

Compilation

- **Optimizer**: Adam with a learning rate of `learning_rate` (default: 0.001).
- **Loss Function**: Sparse Categorical Crossentropy.
- **Metrics**: Accuracy and Mean Squared Error.

Usage

1. **Model Creation**: Call `create_mnist_model()` to create an instance of the model. Modify default parameters as needed.
2. **Training**: Train the model using the normalized MNIST training data.
3. **Evaluation**: Evaluate the model's performance using the MNIST test dataset.

Saving the Model

- The model is saved as 'mnist_model.keras' after creation.

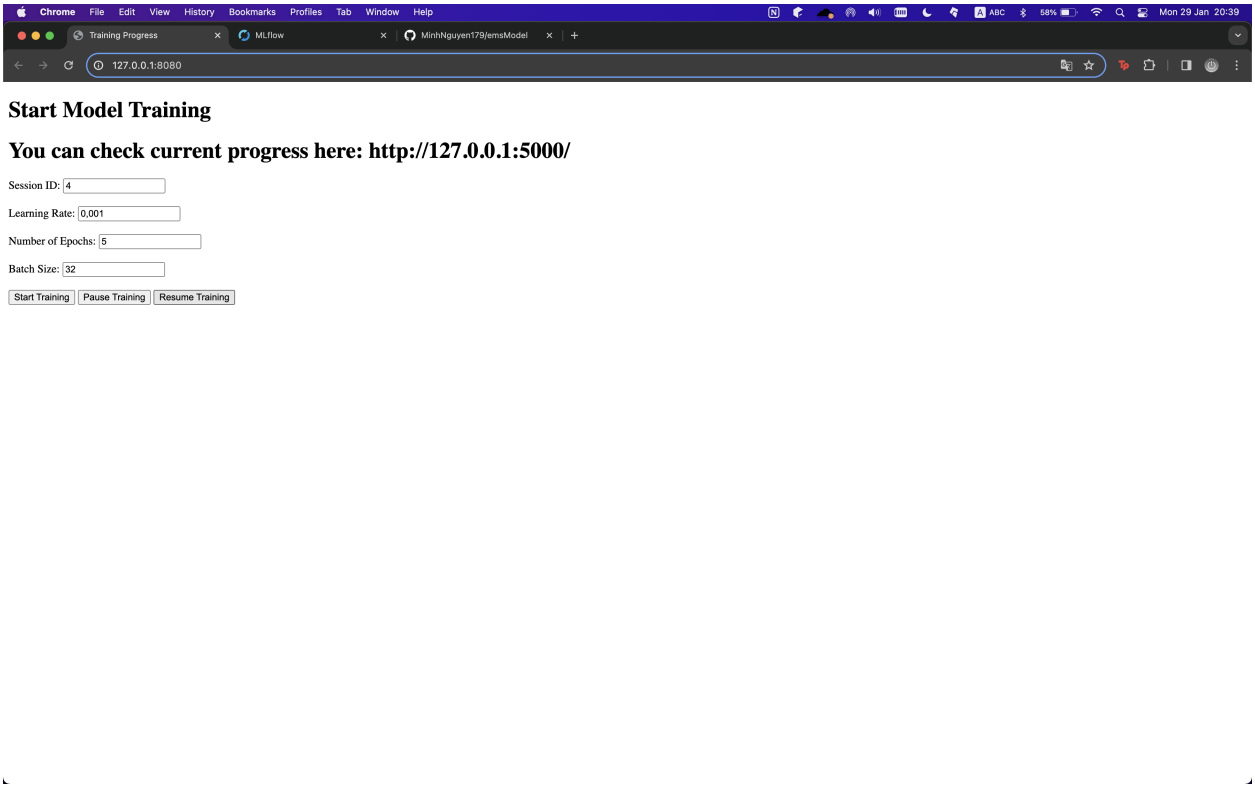
Customization

- Users can customize the number of filters, kernel size, number of dense layer units, and learning rate by passing parameters to `create_mnist_model()`.

It can be expanded with additional sections such as 'Performance Expectations', 'Troubleshooting', or 'Advanced Customizations' based on the user's needs.

Experiment Management System

Draft Interface for Model Management System



Overview

This part provides instructions and details for the Model Training Interface, a web-based application designed to manage the training of machine learning models. This interface allows users to initiate model training sessions with customizable hyperparameters, as well as to pause and resume these sessions as required. Additionally, it provides a link to monitor current training progress.

Key Features

1. **Start Training**: Initiate a new training job with the specified hyperparameters.
2. **Pause Training**: Temporarily halt the ongoing training job.
3. **Resume Training**: Continue a paused training job from where it was left off.

4. **Real-Time Progress Display**: A link is provided to view the current running epoch and overall progress of the training session.

Interface Elements

- **Session ID**: A unique identifier for each training session. Users can start or resume a job corresponding to a given Session ID.
- **Learning Rate**: The learning rate hyperparameter for the training algorithm.
- **Number of Epochs**: The total number of epochs over which the model will be trained.
- **Batch Size**: The number of samples processed before the model is updated.
- **Start Training Button**: Commences the training process with the given hyperparameters.
- **Pause Training Button**: Pauses the ongoing training session.
- **Resume Training Button**: Resumes a paused training session.

Usage Instructions

1. **To Start a New Training Session**:
    - Enter the desired hyperparameters (Learning Rate, Number of Epochs, Batch Size).
    - Click on the 'Start Training' button.
2. **To Pause an Ongoing Training Session**:
    - Click on the 'Pause Training' button.
3. **To Resume a Paused Training Session**:
    - Make sure the correct Session ID is selected.
    - Click on the 'Resume Training' button.
4. **To Monitor Training Progress**:
    - Click on the provided link (e.g., http://127.0.0.1:5000/) to view the current running epoch and training progress.

Monitoring Training Progress

The training progress can be monitored by visiting the provided URL, which is accessible via the dashboard. The monitoring page typically includes detailed information such as current epoch, loss, accuracy, and validation metrics, visualized through graphs or logs that update in real time.

Advanced Usage

- **Concurrent Training**: Users can manage multiple training sessions by using different Session IDs. This allows for parallel model training and experimentation with various hyperparameters.
- **Session Management**: Each Session ID corresponds to a unique instance of a training job, ensuring that multiple training processes can be independently managed without interference.

Notes

- Ensure that the training session's ID is correctly noted for pausing or resuming the session later.
- The interface does not automatically save the model's state after pausing. Users should ensure that their training scripts include functionality to save and load the model's weights as needed.

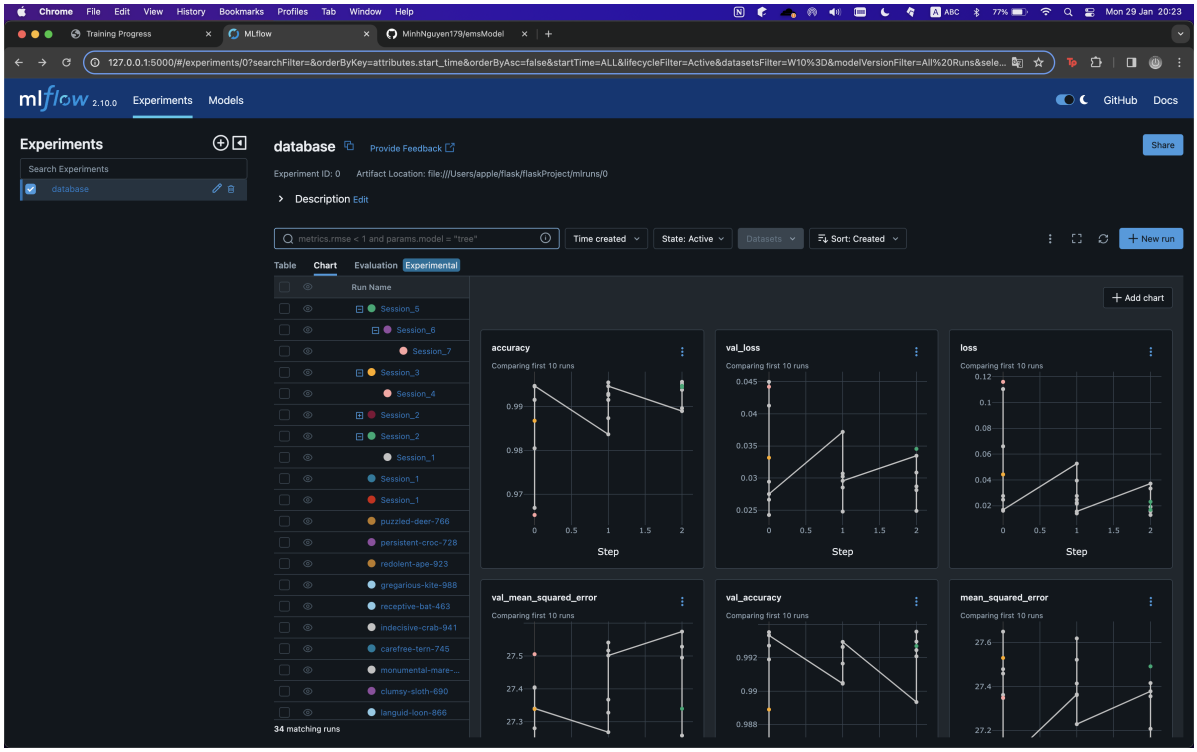
Troubleshooting

- **Session Does Not Resume**: Verify that the correct Session ID is selected and that the server is running without errors.
- **No Progress Display**: If the progress link does not show the current status, check the server status and ensure the monitoring service is operational.

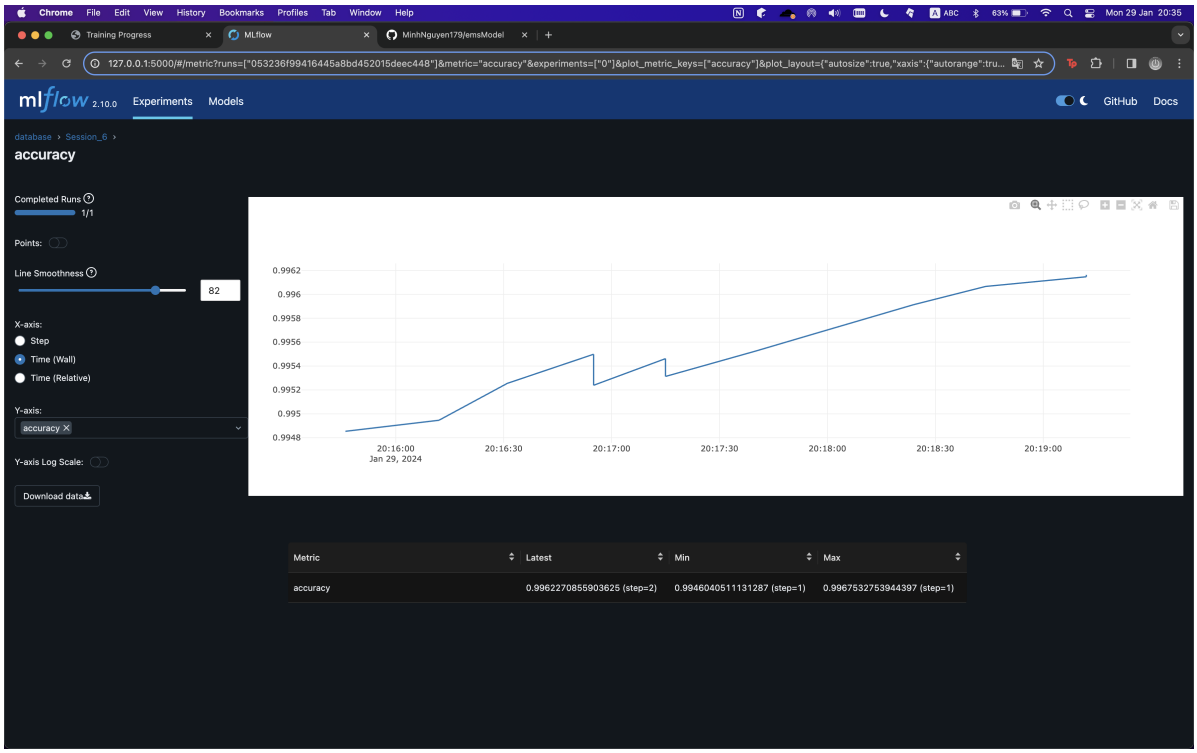
Advanced Management Dashboard for Concurrent Training Sessions

The management dashboard is meticulously designed to facilitate multiple concurrent training sessions, enhancing the efficiency and depth of model analysis. In here I apply MLFlow for building a dashboard to keep track all the current process with database to log all informations of each job. Key features include:

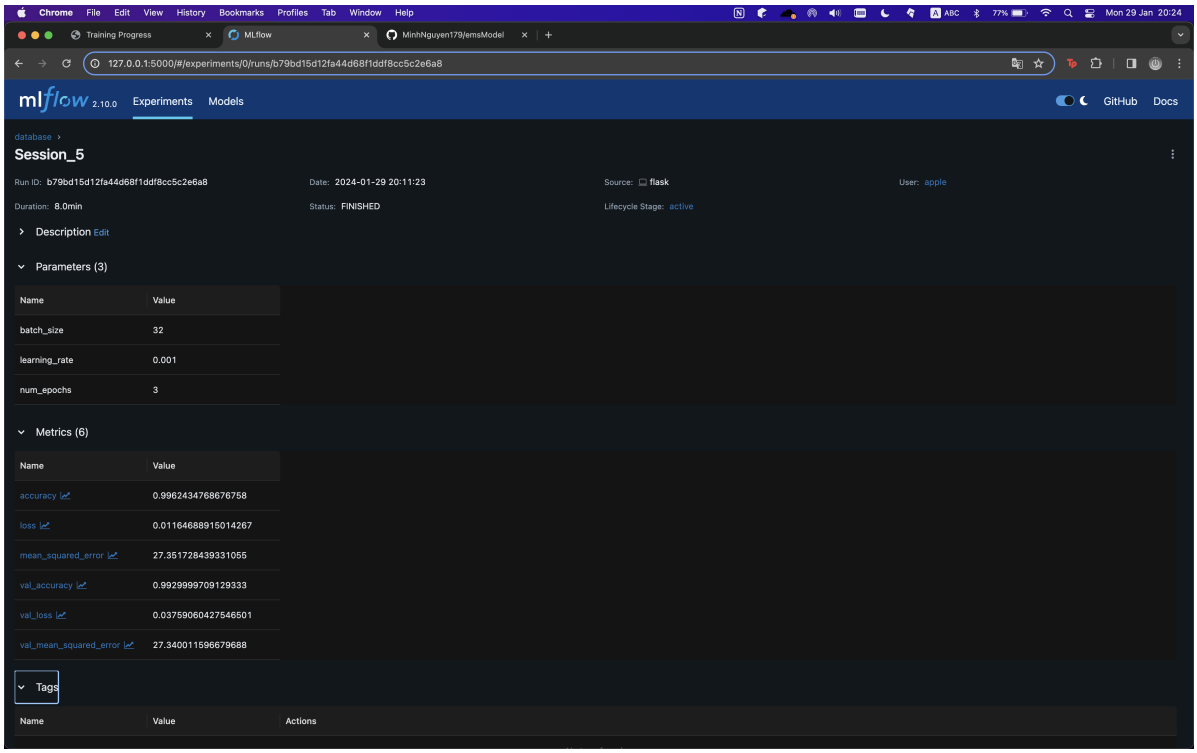
1. **Nested Training Threads**: The dashboard's foremost feature is its capability to run nested training threads simultaneously. This innovative approach allows users to execute multiple training jobs concurrently, significantly streamlining the model development process.



2. **Enhanced User Experience**: Users can initiate, monitor, and manage multiple jobs simultaneously, providing a seamless and intuitive experience. This multi-threaded environment is especially beneficial for complex analyses involving comparative studies across different model configurations or datasets.
3. **Deep Model Analysis**: By running multiple training sessions in parallel, users can conduct a comprehensive and in-depth analysis of the model. This feature is particularly advantageous for experimenting with various hyperparameters, architectures, and training strategies, offering real-time insights into each model's performance.
4. **Comparative Result Analysis**: The dashboard is equipped with sophisticated tools for comparing results from different training jobs. This comparative analysis is crucial for identifying optimal model configurations and understanding the impact of various training parameters.



5. **Real-Time Monitoring and Control**: Users have real-time access to each training session's progress, with capabilities to adjust parameters, pause, or terminate sessions as needed. This level of control is vital for responsive model tuning and quick iteration cycles.



6. **Data Visualization and Reporting**: The dashboard includes advanced data visualization tools and reporting features. These tools enable users to visualize training progress, compare metrics across different sessions, and generate detailed reports for in-depth analysis.