Nitish Bhardwaj (B21AI056)
MLOps Assignment 6
Quantum Machine Learning

Codebase: https://colab.research.google.com/drive/1LZylrY_aOijBefEIiCQn-TdDiz-ZynZV?usp=sharing
MLFlow Report: https://dagshub.com/Niticodersh/Mlops_assignment.mlflow/#/experiments/2

The primary objective was to build a machine learning model using the digits dataset, train it, and implement model quantization to improve efficiency without significantly compromising accuracy.

Environment Setup
The assignment utilized the following libraries:

PyTorch: For building and training the machine learning model.
MLFlow: For tracking experiments, logging metrics, and managing models.
DagsHub: For repository management and integration with MLFlow.
The project environment was set up with necessary package installations, and MLFlow was initialized to manage experiments efficiently.

Data Loading
The digits dataset was loaded using load_digits() from Scikit-learn. The data was then standardized using StandardScaler, followed by splitting into training and testing datasets (80% training, 20% testing).

Model Building
A logistic regression model was defined using PyTorch's nn.Module. The model architecture consisted of a single linear layer to map input features to output classes.

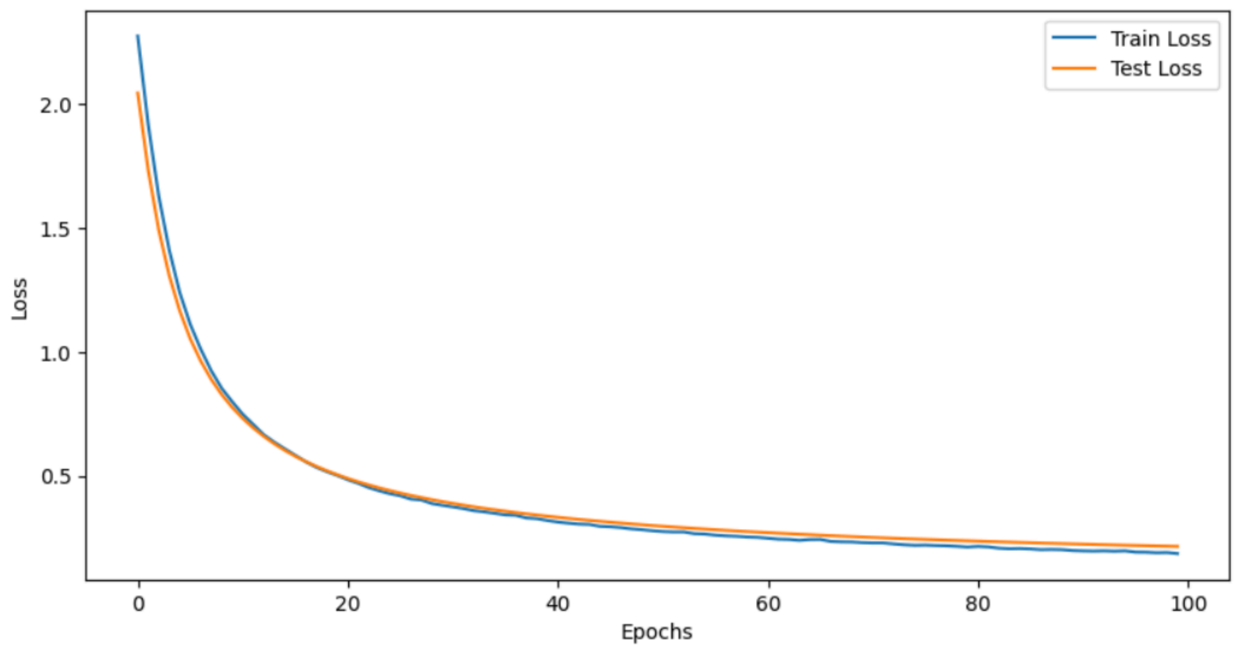
Softmax Handling
During training, the softmax activation function was not explicitly applied, as the Cross-Entropy Loss function incorporates it internally. This allows for a more stable computation of the loss directly from the raw logits produced by the model. The softmax function was applied during testing to convert the model outputs into probabilities for class predictions, enabling accurate evaluation of the model's performance.

Data Transformation
The training and testing data were converted into PyTorch tensors. A DataLoader was created to facilitate batching during training.

Model Training
The model was trained over 100 epochs using the following configurations:

Loss Function: Cross-Entropy Loss
Optimizer: Stochastic Gradient Descent (SGD) with a learning rate of 0.01

During training, both training and testing losses were logged for each epoch using MLFlow. The best model (with the lowest testing loss) was saved for future evaluation.



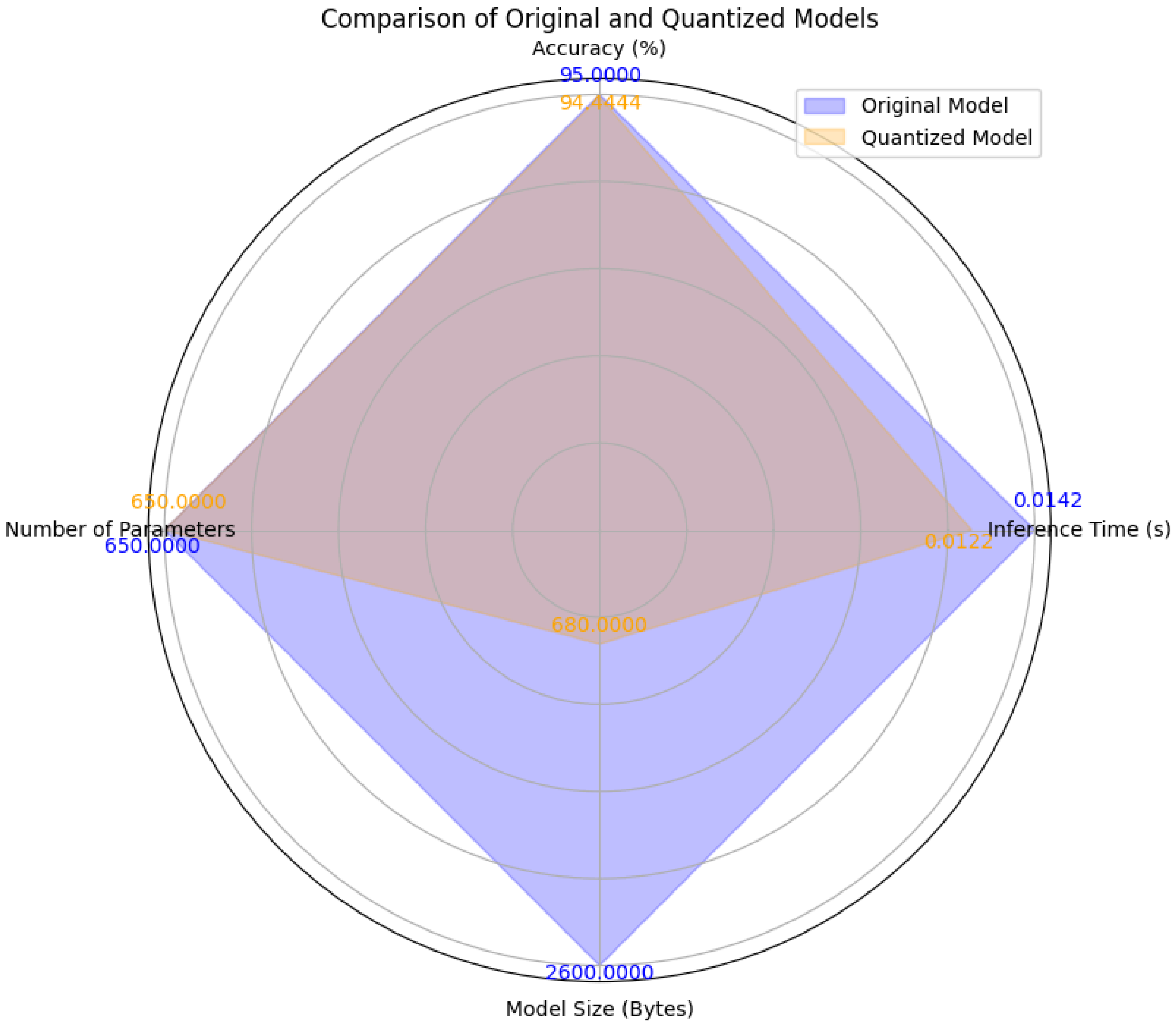
Model Inferencing
After training, the model's performance was evaluated on the test dataset. Inference time and accuracy were recorded, with results logged in MLFlow for further analysis.

Model Quantization
To optimize the model for deployment, dynamic quantization was applied. This involved converting the weights of the model to lower precision (from float32 to int8), thereby reducing the model's size and improving inference speed.

Model Comparison
The performance of the original and quantized models was compared based on:
<u>Original Model</u>
Inference Time: 0.0142 seconds
Accuracy: 95.0000%
Number of model parameters: 650
Model size: 2600 Bytes
We can see number of params = 650, which makes size of model = 650 * 4 = 2600, since type is float32 which takes 32 bits or 4 Bytes

<u>Quantized Model</u>
Inference Time: 0.0122 seconds
Accuracy: 94.4444%
Number of model parameters: 650
Model size: 680 Bytes

We can see the model size is 680 bytes because in pytorch it quantizes only weights and not biases. Weights param = 640 and bias params = 10. Therefore, model size = 640*1 + 10*4 = 680 Bytes



The results showed that the quantized model achieved comparable accuracy while significantly reducing the model size and inference time.