PROBLEM 2

Explanation of how the dataset was created:

The dataset was created by using the latest 3 days' data as the features and the next day's opening price as the target. Each row in the dataset corresponds to one sample instance. The features for each sample consist of the opening prices for the three previous days, while the target is the opening price of the following day.

Any preprocessing steps followed:

Before training the neural network, some preprocessing steps were performed on the dataset:

1. Feature Scaling: Min-max scaling was applied to scale all the features and the target variable within the range of [0, 1]. This step is essential to ensure that all the values have a similar scale, which helps in faster convergence during training.

2. Reshaping for LSTM: LSTM networks require input in a 3D format [samples, time steps, features]. The features were reshaped to have a single time step for each sample.

All design steps to find the best network and final design:

The design steps include:
1. Initializing a sequential model.
2. Adding an LSTM layer with 50 units and a ReLU activation function. The input shape is set to (1, n), where n is the number of features.
3. Adding a Dense layer with 1 unit to output the predicted value.
4. Compiling the model using the Adam optimizer and mean squared error loss function.

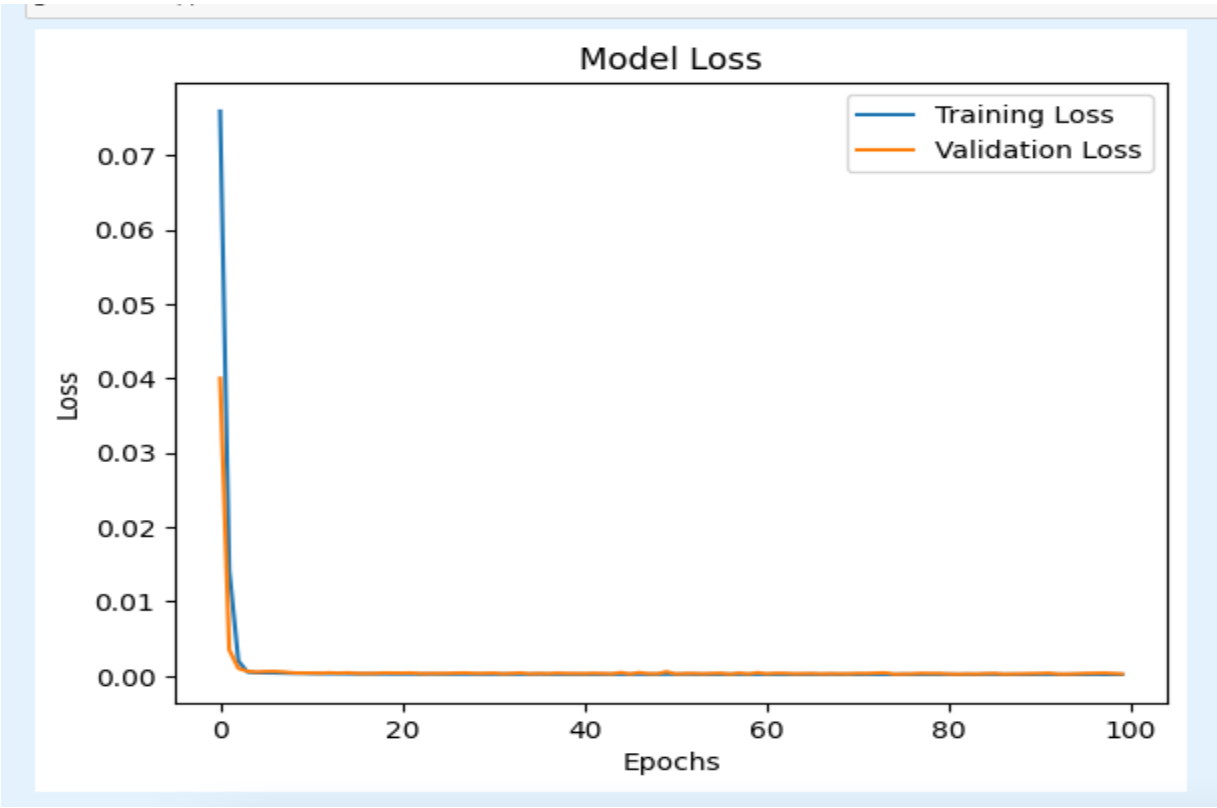
Architecture of the final network, number of epochs, batch size, loss function, training algorithm, etc.:

The final network architecture is as follows:
- LSTM layer with 50 units and ReLU activation.
- Dense layer with 1 unit.

Number of epochs: 100
Batch size: 32
Loss function: Mean squared error (MSE)
Optimizer: Adam

Output of the training loop with comments on your output:

The model is trained on the training data for 100 epochs using a batch size of 32. During training, the ModelCheckpoint callback is used to save the best model based on validation loss. The final training loss is printed at the end of the training loop.



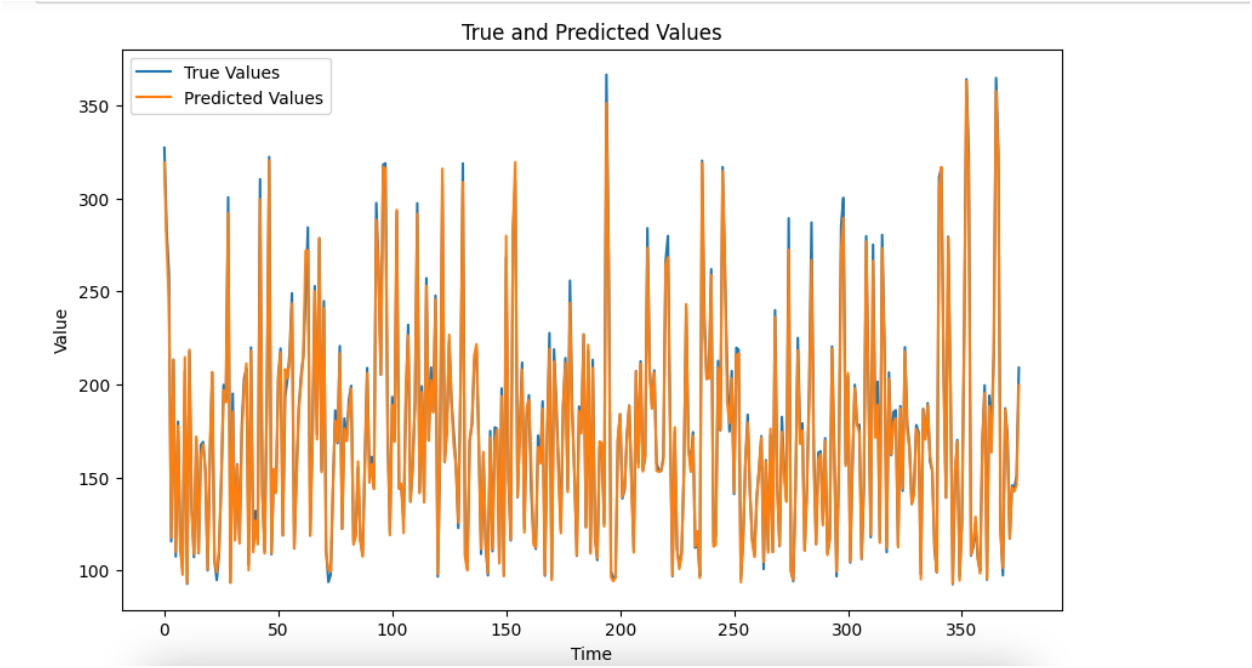
From the provided figure, it is evident that as the number of epochs increases, the loss gradually decreases. Initially, the decrease is rapid, and it almost converges at a training loss of approximately 25 and a validation loss of around 20.

The final training loss is impressively low at 0.0001937259512487799, which is likely achievable due to the use of the Adam optimizer. Unlike classical stochastic gradient descent, the Adam optimizer dynamically updates the learning rate during the training phase. This adaptive learning rate helps the model efficiently navigate the loss landscape and converge to a better solution.

In summary, the observed decrease in loss with increasing epochs, especially the low final training loss, can be attributed to the benefits of utilizing the Adam optimizer, which facilitates effective learning rate adjustments during the training process.

Output from testing, including the final plot and your comment on it:

The trained model is loaded, and predictions are made on the test data. The test loss is calculated and printed. A plot is generated to compare the true opening prices from the test data with the predicted opening prices from the model. The plot shows how well the model's predictions align with the true values.



Upon loading the saved model and making predictions, it is observed that the Mean Squared Error (MSE) test loss is measured at 0.00024488221970386803. Overall, the model seems to perform well, as most of the predicted values closely match the true values from the testing dataset.

However, it is worth noting that there are a few predictions that do not align closely with the actual testing values. These discrepancies might be attributed to various factors, such as noisy or challenging data points, model complexity, or potential overfitting of the training data.

While the majority of the predictions being accurate is a positive sign, the presence of some less accurate predictions indicates that there may still be room for improvement in the model's generalization capability.

Further analysis and fine-tuning of the model, or potentially exploring different architectures, hyperparameters, or regularization techniques, could help address the deviations between predictions and testing values and enhance the overall performance of the model.

What would happen if you used more days for features (feel free to actually try it, but do not upload the datasets)?

If more days were used for features, the model would have access to additional historical data for each sample instance. This may lead to a more comprehensive representation of the data and potentially improve the model's ability to capture underlying patterns and trends.

However, using more days for features may also increase the complexity of the model and make it more susceptible to overfitting, especially if the dataset is limited. It would be important to strike a balance between the number of days used as features and the overall dataset size.

To try using more days for features, we would need to modify the "days" variable in the initial dataset creation code. For example, if we set "days = 5," the model would use the latest 5 days as features and the opening price of the 6th day as the target. We would need to experiment with different values of "days" and evaluate the model's performance on a validation set to find the optimal number of days to use for features.