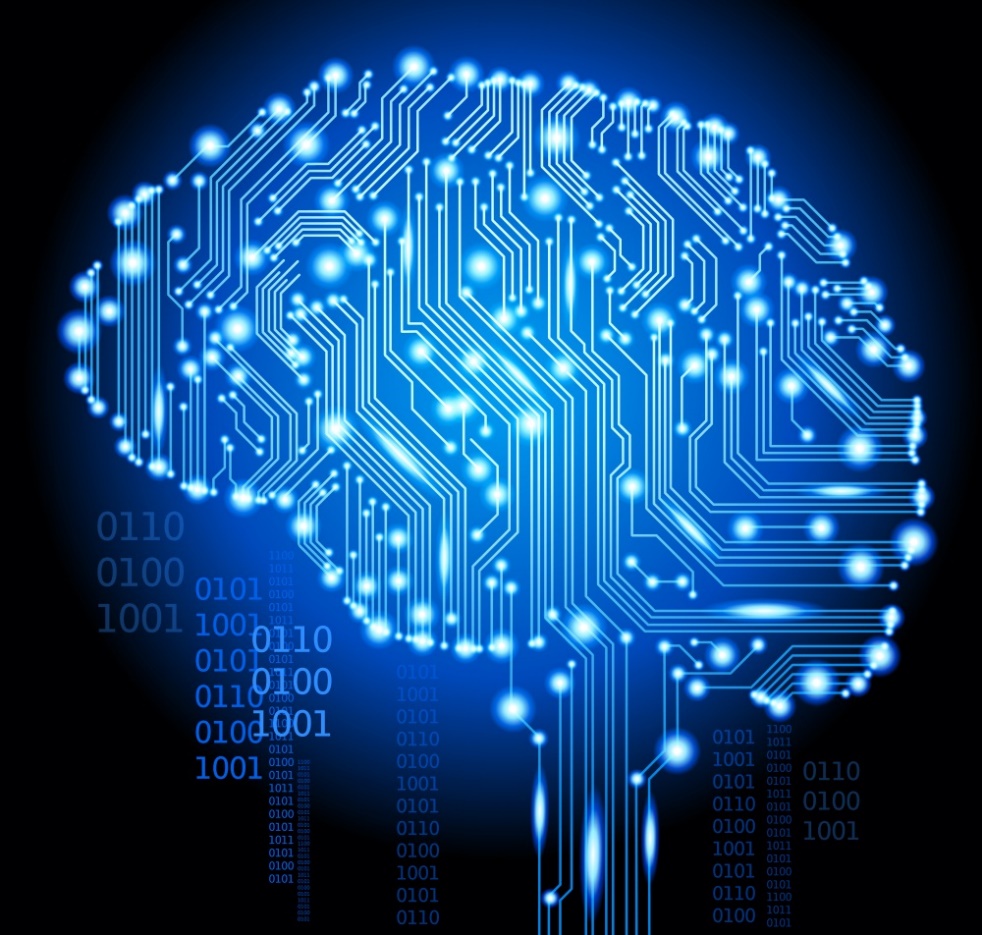


COS30018 – Intelligent Systems

**REPORT**

TASK B4 – MACHINE LEARNING 1



**TABLE OF CONTENTS**

*Table of contents…………………………………………………………………………………………………………………2*

*I. Creating Deep Learning model…………………………………………………………………………………………3*

*II. Testing…………………………………………………………………………………………………………………………….5*

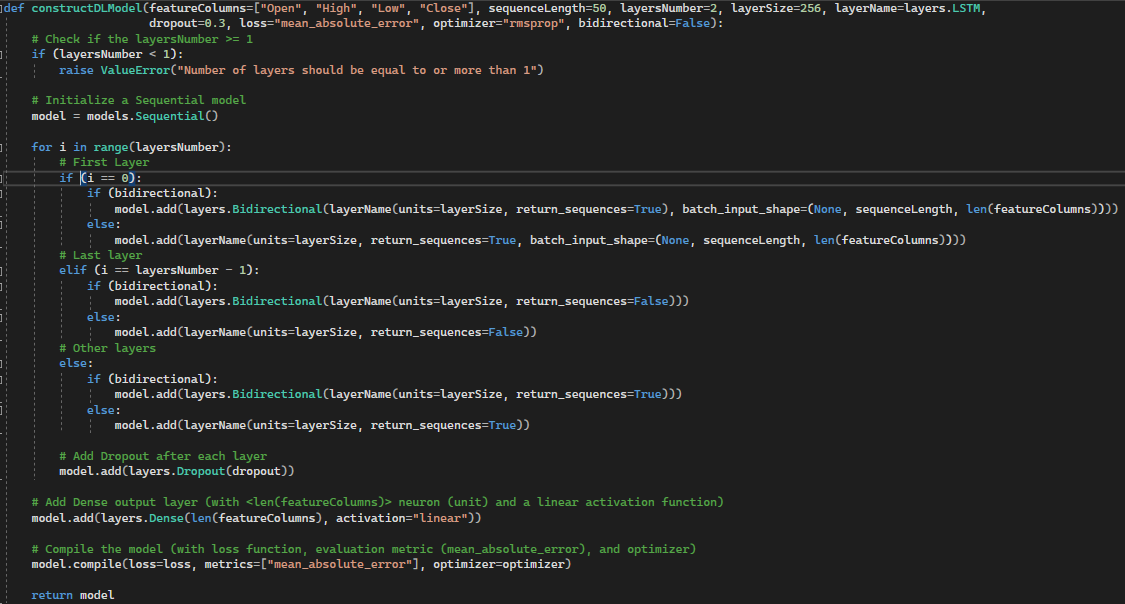
*1. Test with LSTM layer………………………………………………………………………………………………5*

*2. Test with multiple types of price data*

*III. References…………………………………………………………………………………………………………………*

**CREATING DEEP LEARNING MODEL**

For this task, I have create a new file of “machineLearning.py”, with the method “constructDLModel”, taking some parameters including the number of layers, each layer’s size, layer’s name (cell), sequence length, dropout, loss, optimizer and bidirectional, and returning a Deep Learning model.



* “sequenceLength”: The length of input sequences
* “layersNumber”: The number of layers in neural network
* “layerSize”: The number of units in each layer
* “layerName”: The type of layer to use (e.g. LSTM, GRU, RNN, …)
* “dropout”: The dropout rate, a regularization technique to prevent overfitting
* “loss”: The loss function to optimize during training (e.g. “mean\_absolute\_error”, “mean\_squared\_error”, “huber\_loss”, …)
* “optimizer”: The optimization algorithm used during training (e.g. “rmsprop”, “adam”, …)
* “bidirectional”: A boolean variable indicating whether to use bidirectional layers.

Initially, the method checks whether the specified number of layers is less than 1. If so, it raises a ValueError, indicating that this number should be at least 1.

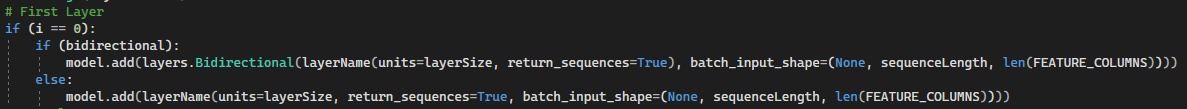


The next stage is the main process of model creating. First, a Squencial model is initialized, which will then be used to build the neural network.

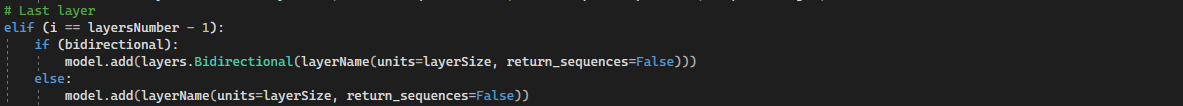


Next, a loop is implemented to add layers to the model, running from 0 to “layersNumber” – 1, allowing to specify the number of layers in the model:

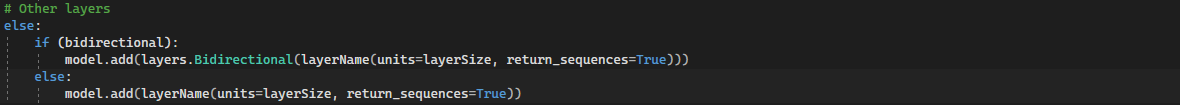
* If the first layer is added: If the bidirectional processing is enable (“bidirectional” = false), a Bidirectional layer is added along with the specified “layerName” and other configurations from the parameters, otherwise a standard layer of “layerName” is added only. For the first layer, in both cases, the “return\_sequences” is True, and the “batch\_input\_shape” is configured with the same tuple: (None, “sequenceLength”, len(FEATURE\_COLUMNS) (number of considered columns of price values)).



* If the last layer is added: Similar for both cases of “bidirectional” of the first layer case, but the “batch\_input\_shape” is not configured, while the “return\_sequences” parameter will be False as there would not be no layer added further.



* If the added layer is between 2 and “layersNumber” – 2: These layer addings are similar to the last layer adding, except the True value of the “return\_sequences” parameter.



Following that, after each layer adding, including the last one, a Dropout layer is added. This is a regularization technique that helps prevent overfitting by randomly dropping a fraction of the neurons during training.



After the loop is over, which means all layers are added to the model, a Dense output layer with number of units/neurons equal to the number of feature columns (len(featureColumns) and a linear activation function is added. This is typically used for regression tasks where the model predicts a continuous numerical value:



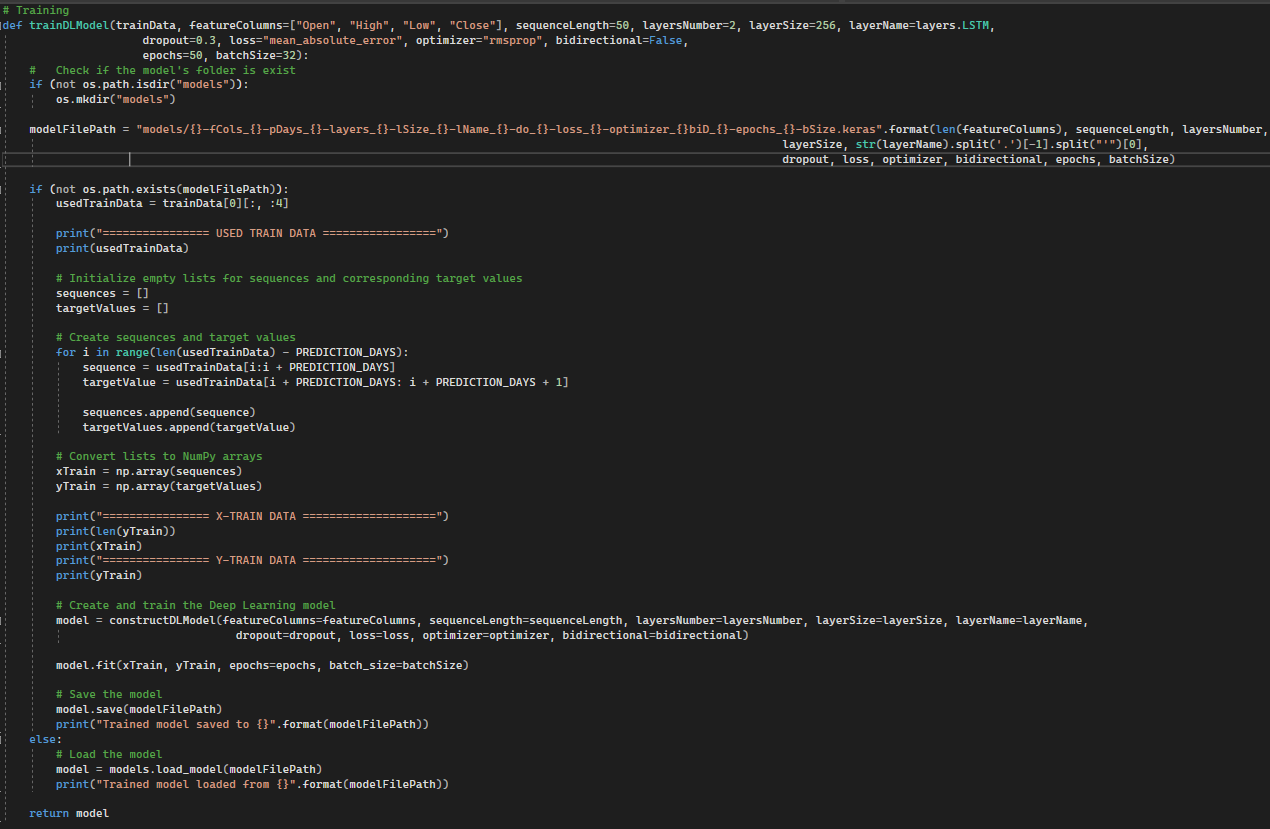
Finally, the model is compiled with the specified loss function, evaluation metric (mean\_absolute\_error), and optimizer, preparing the model for training process before returns it.



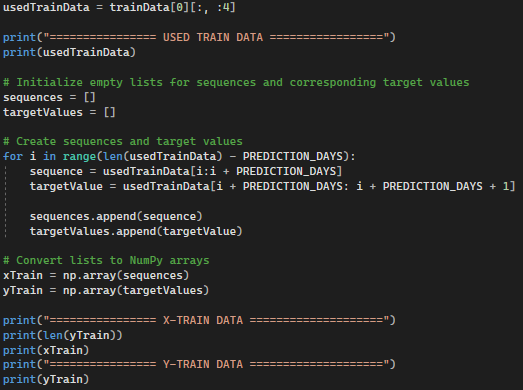
* “mean\_absolute\_error”: This is a common loss function used in regression problems, mostly the issues involving time series predicting. It measures the average absolute difference between the predicted values and the actual target values.
* “rmsprop”, stading for Root Mean Square Propagation, is an optimization algorithm, adjusting the learning rate for each parameter during training based on the past history of gradients.

**TESTING**

In the first phase of creating and training the model, I created another method of “trainDLModel()” for this stage:



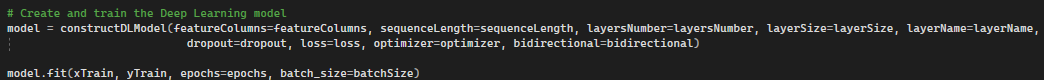
When the matched model does not exist in the “models” folder, the method firstly prepares the training data:



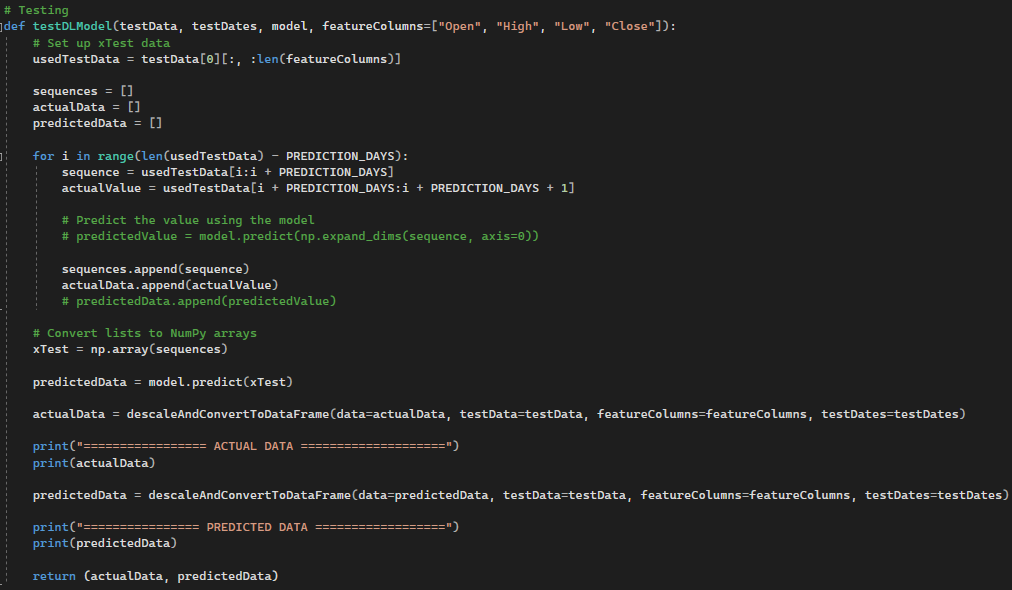
It extracts the first four columns ("Open," "High," "Low," and "Close") from the “trainData” and then creates sequences and corresponding target values for training.

The loop over the training data is implemented, creating sequences of <PREDICTION\_DAYS> and the corresponding target values. These sequences and targets are stored in NumPy arrays “xTrain” and “yTrain”.

Then, I use the “constructDLModel()” method to create the model, and use “model.fit()” method to start the training:

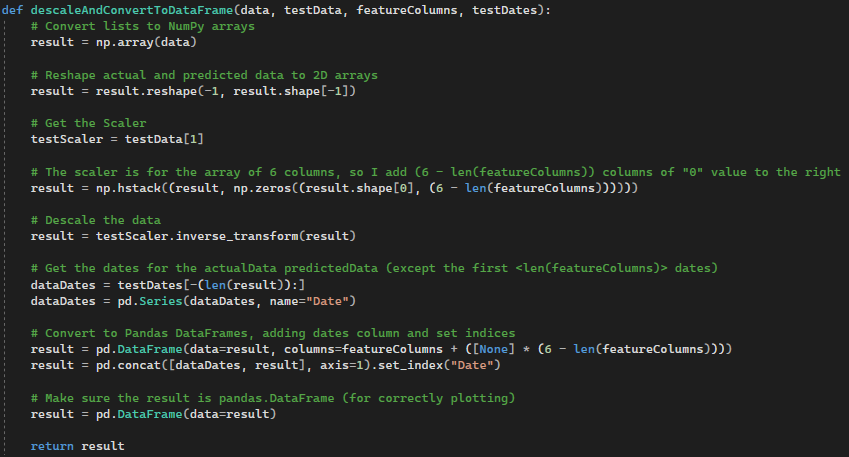


For the second stage, testing the model, the method of “testDLModel()” was created:



It prepares the testData similarly to the previous stage. In this step, sequence values (“xTest”) are paramater of “predict()” function of the model, and the output will be the “predictedData” array.

Both the “actualData” and “predictedData” then go through the converting process, using the “descaleAndConvertToDataFrame”, to be descaled and converted to pandas.DataFrame format, preparing for plotting.

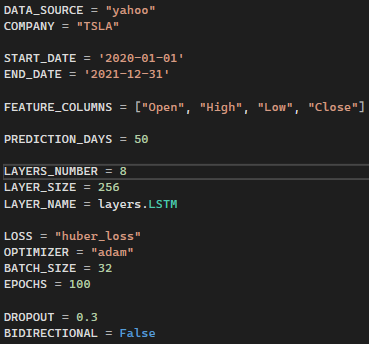


The last one is the plotting process. I chose the simple line graph containing two lines of two preferred feature columns from each array:

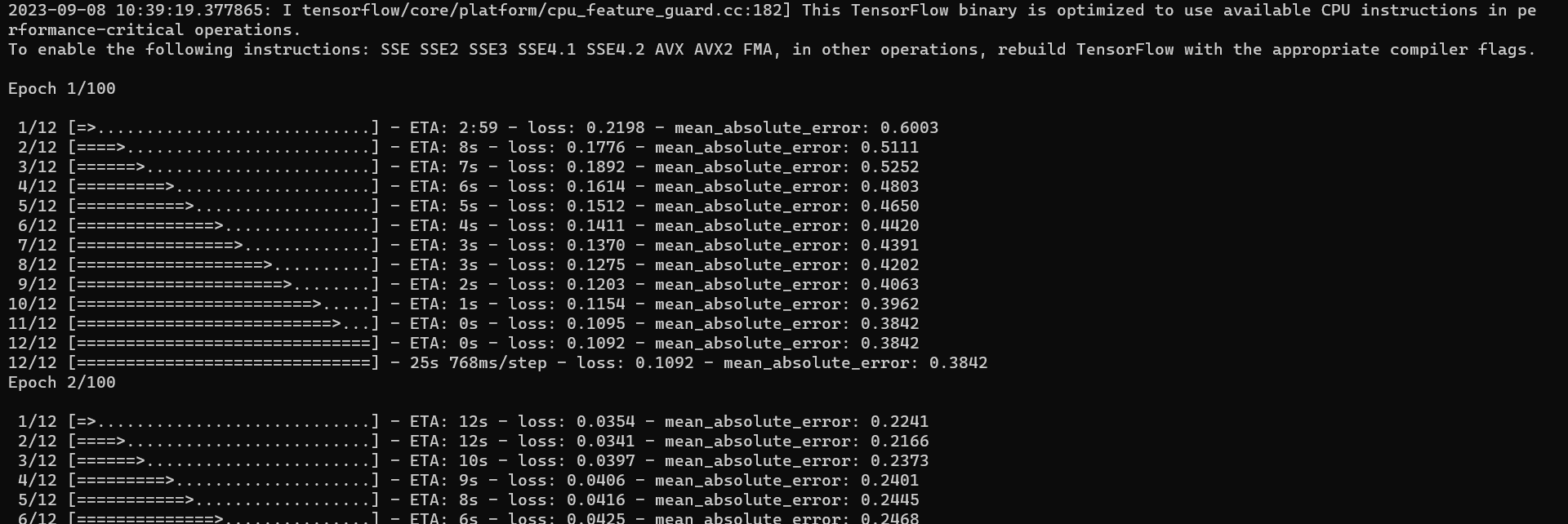


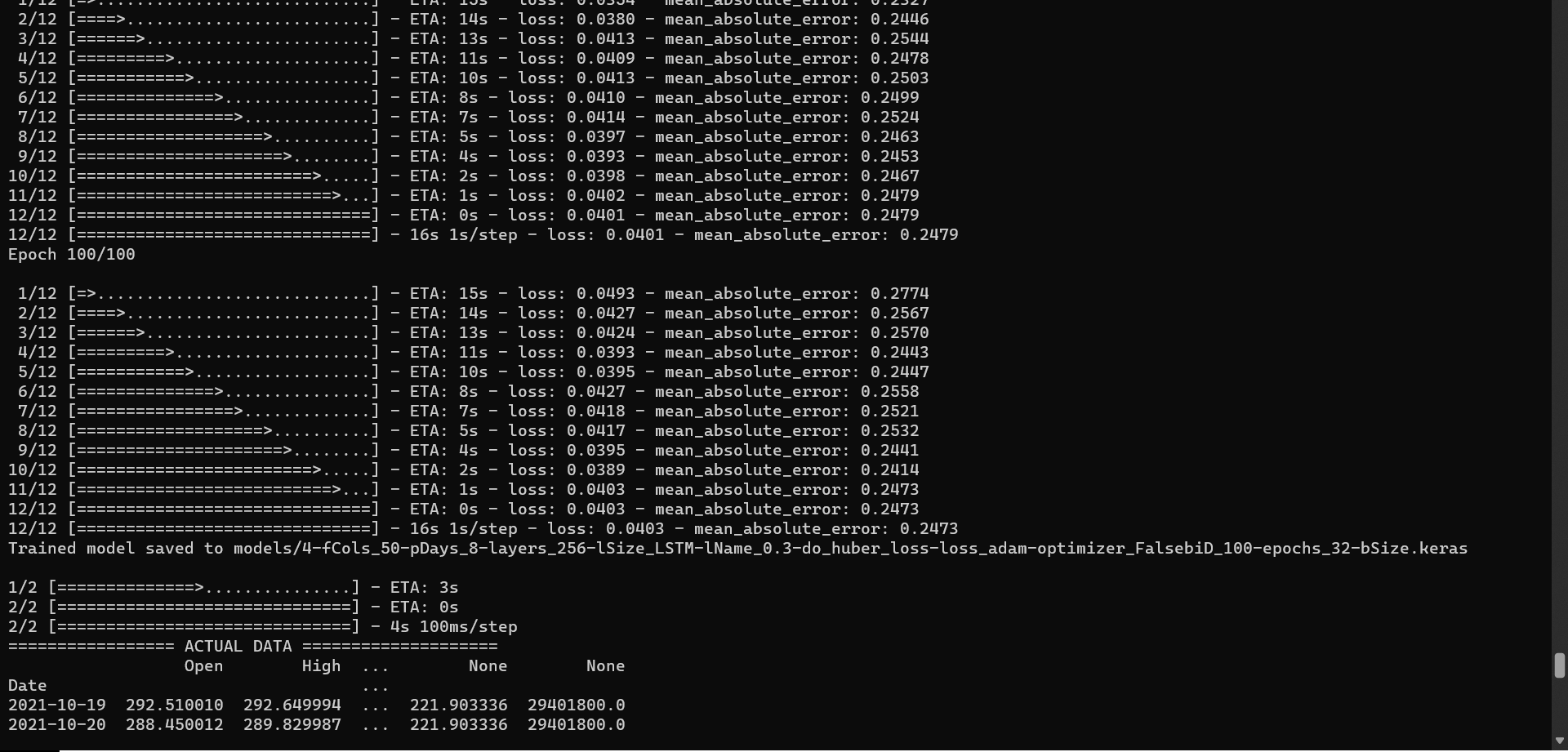
**1. Test with LSTM layer**

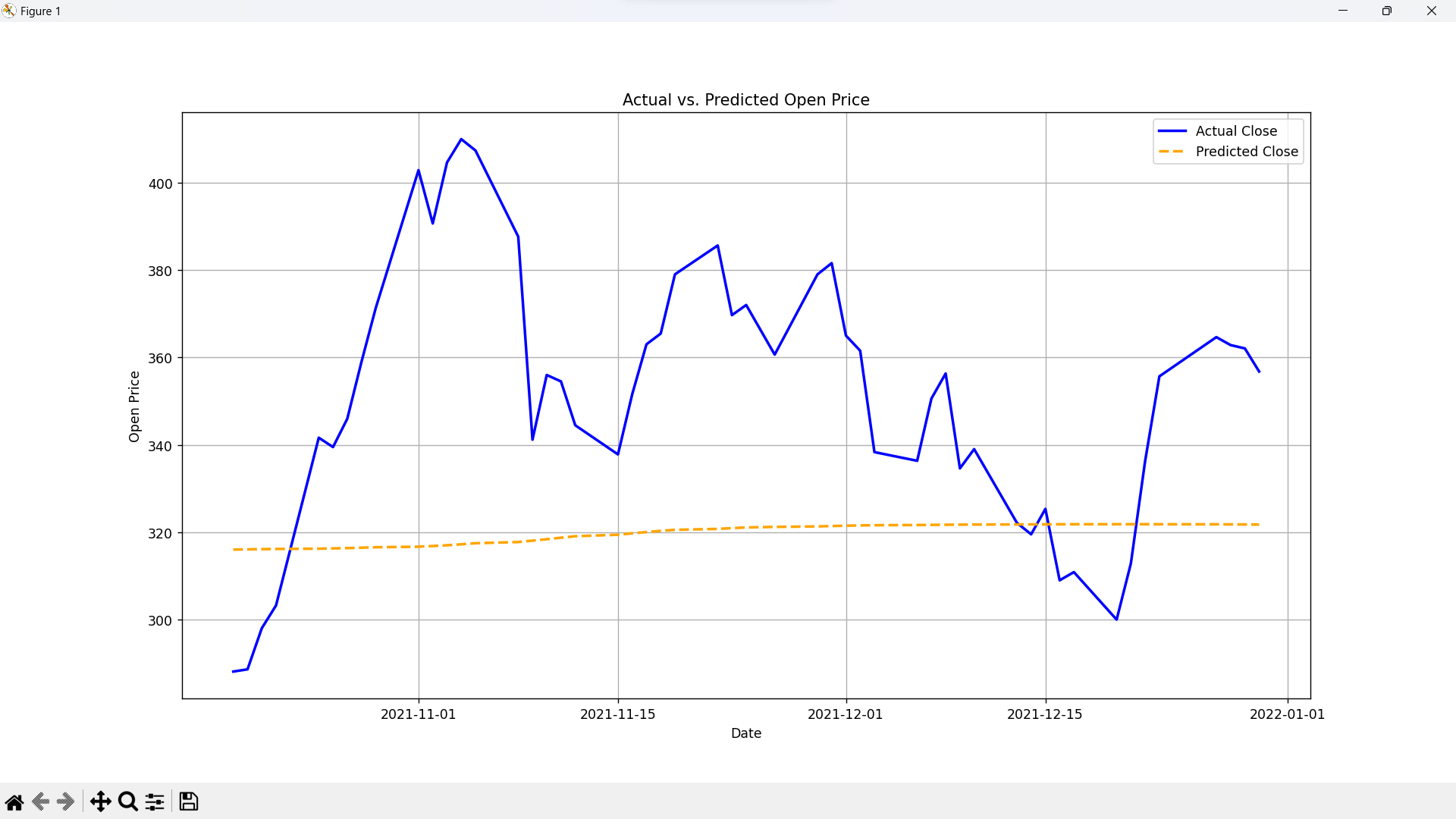
For the first test, some parameter I have used for this test:



Output:

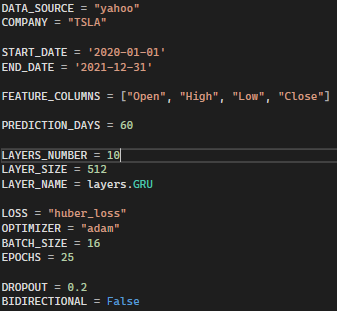




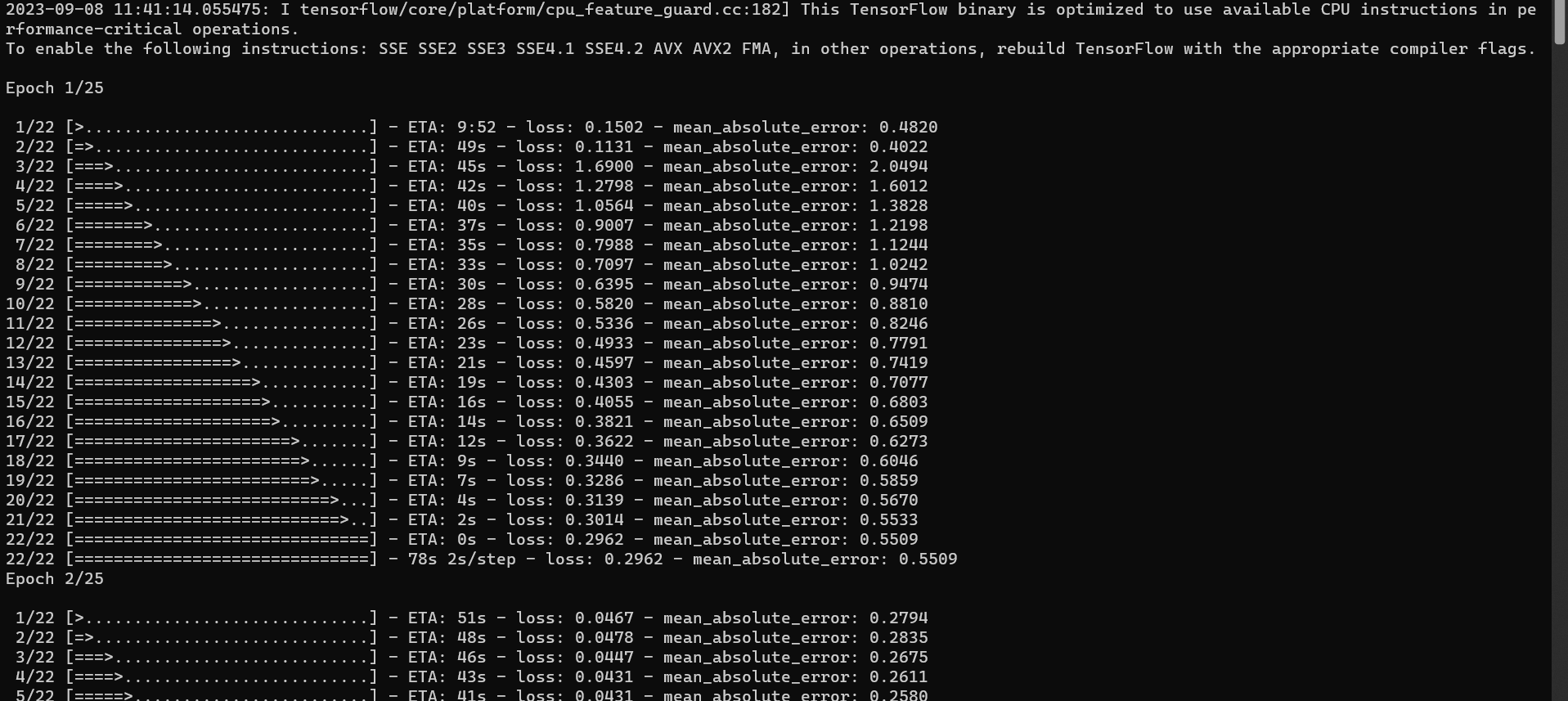


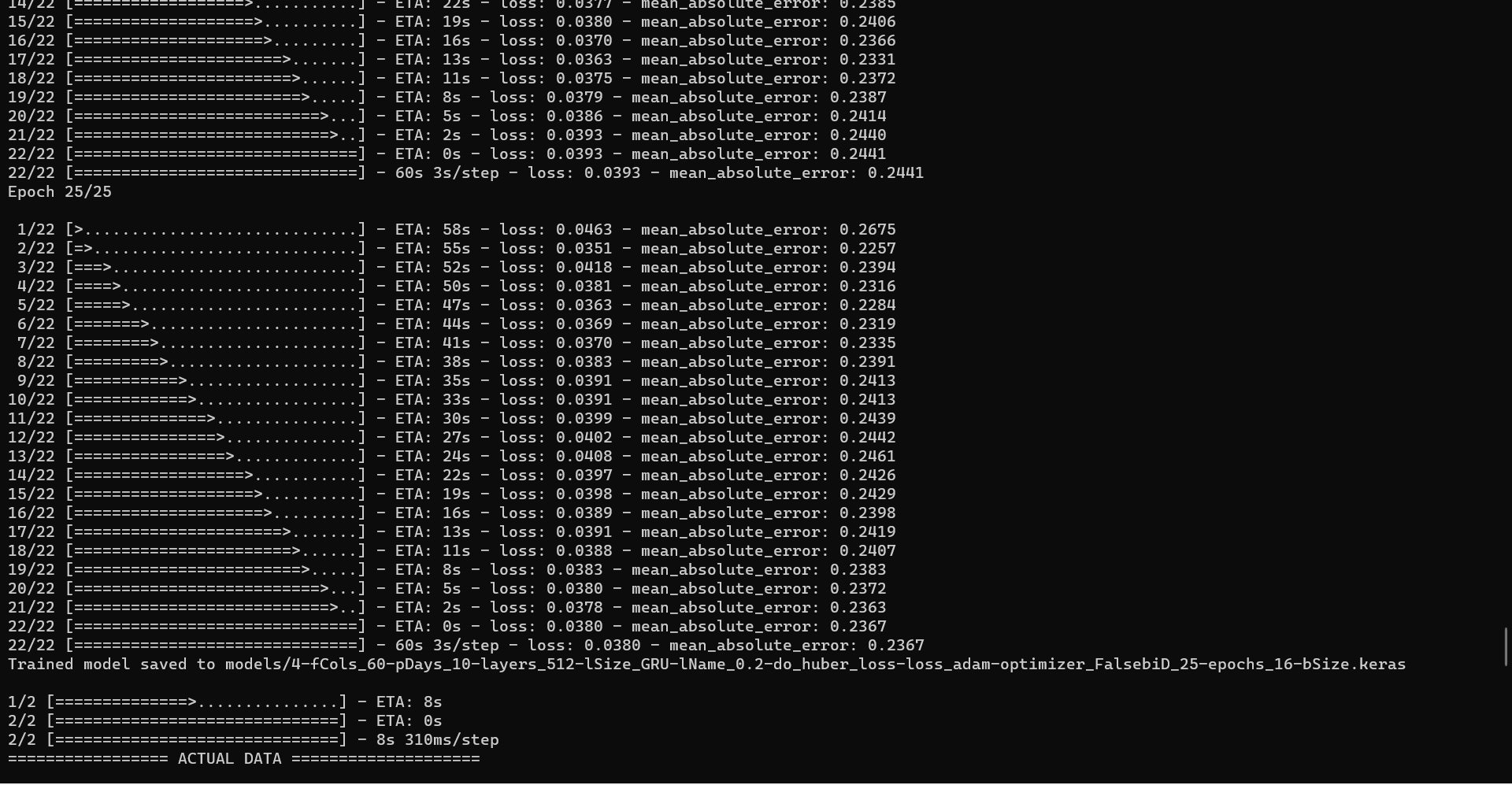
**2. Test with GRU layer**

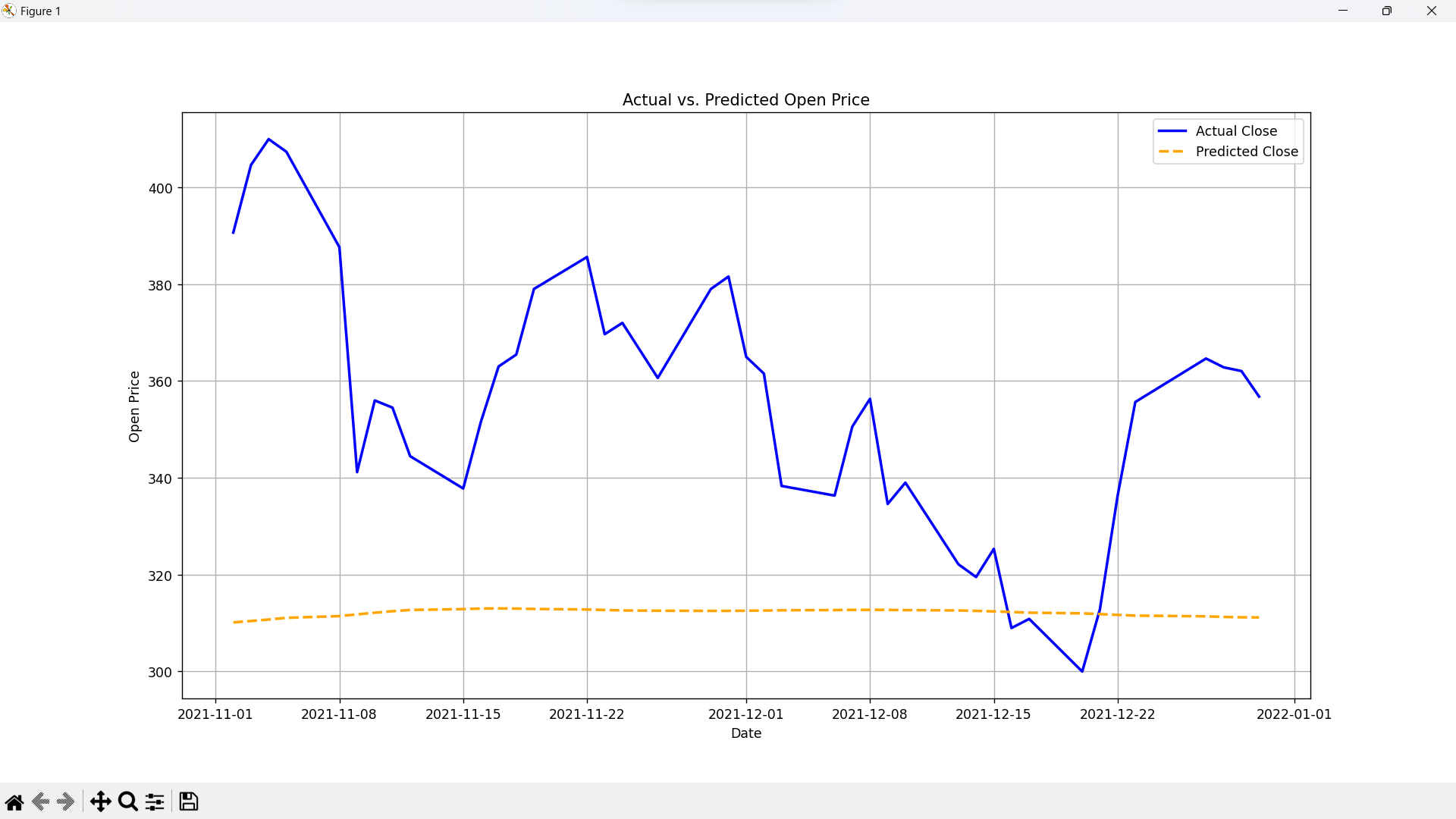
For the second test, some parameter I have used for this test:



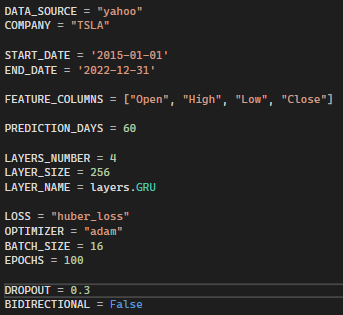
Output:



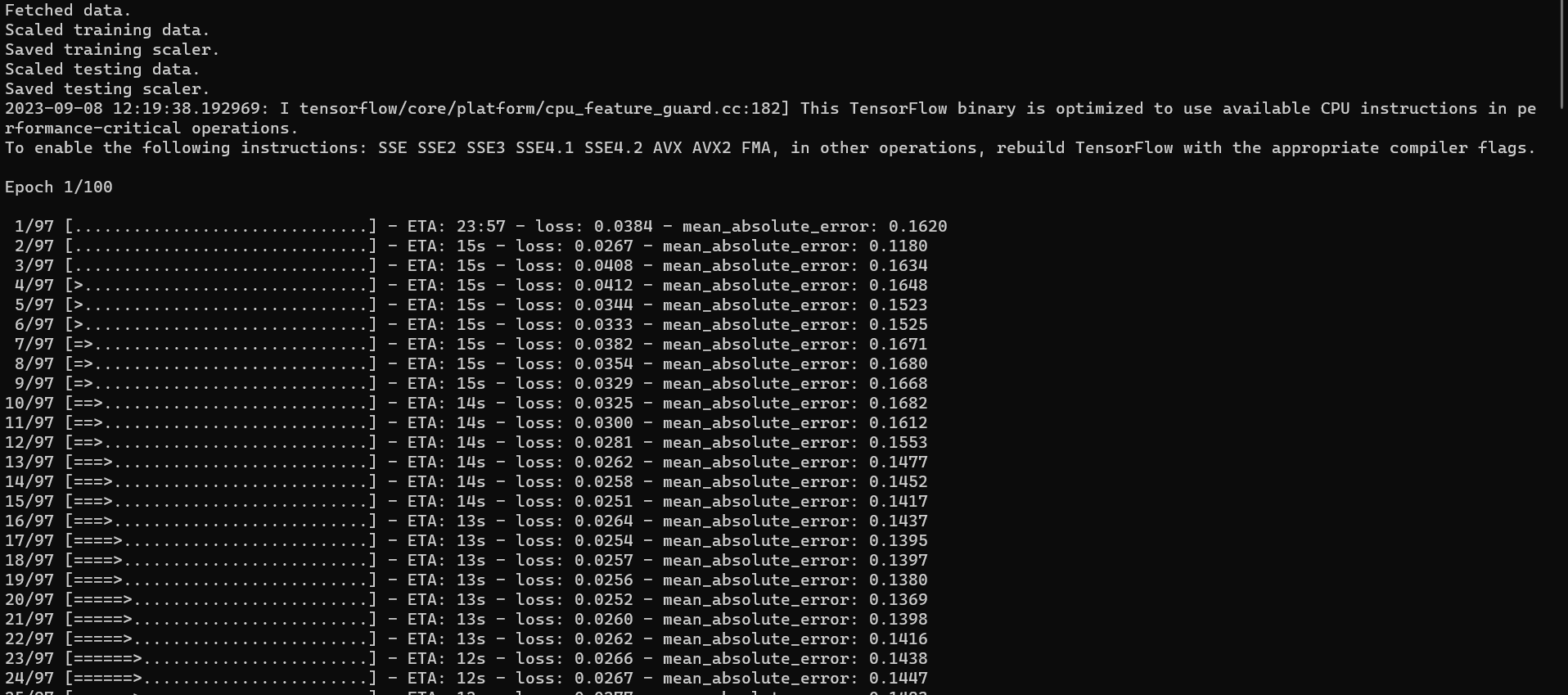




For the third test, some parameters I have used for this test:



Output:



**REFERENCES**