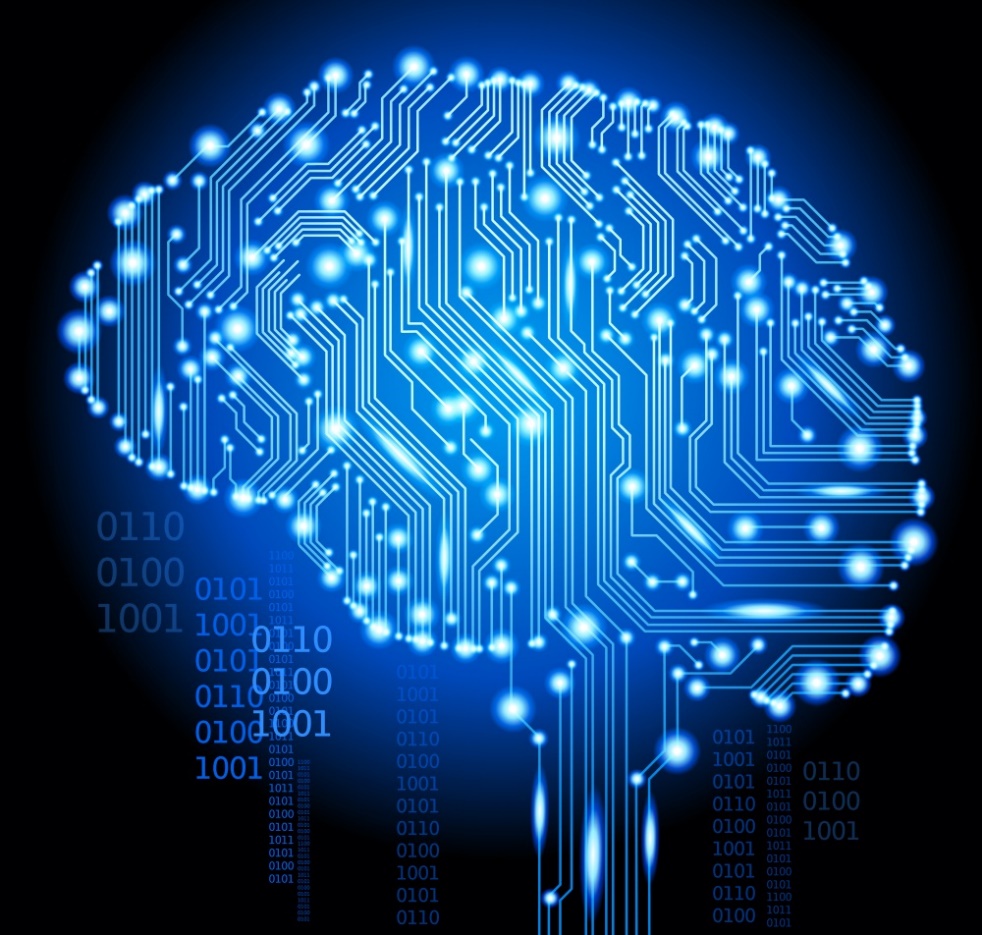


COS30018 – Intelligent Systems

**REPORT**

TASK B4 – MACHINE LEARNING 1



**TABLE OF CONTENTS**

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

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

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

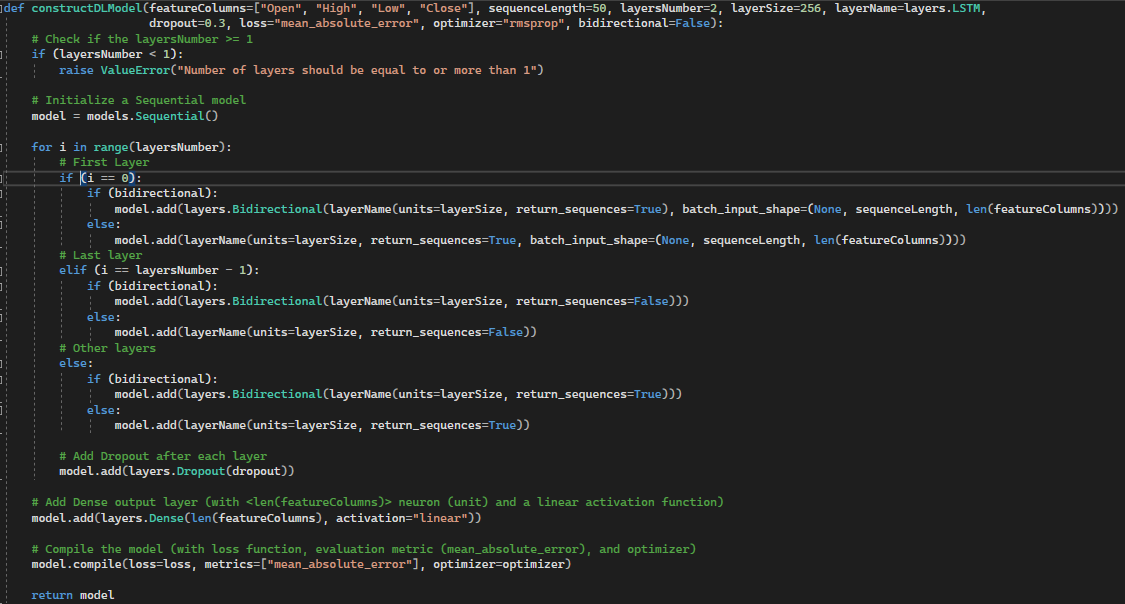
*1. Test with LSTM layer………………………………………………………………………………………………9*

*2. Test with RNN layer………………………………………………………………………………………………11*

*3. Test with GRU layer………………………………………………………………………………………..*

**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.

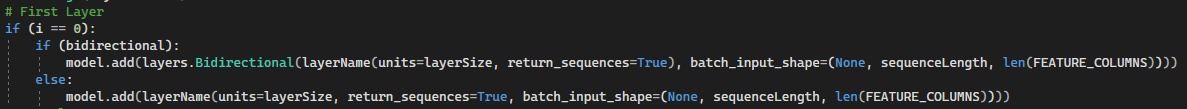


The next stage is the main process of model creating. First, a Squencial model is initialized, which will then be used to build the neural network.

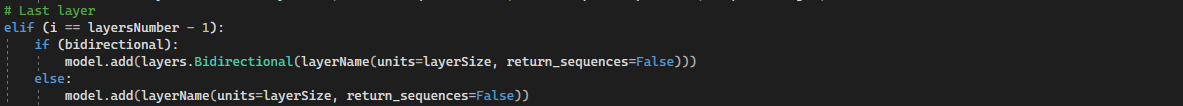


Next, a loop is implemented to add layers to the model, running from 0 to “layersNumber” – 1, allowing to specify the number of layers in the model:

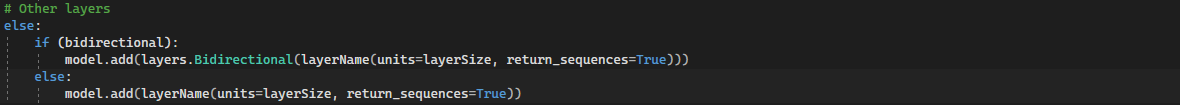
* If the first layer is added: If the bidirectional processing is enable (“bidirectional” = false), a Bidirectional layer is added along with the specified “layerName” and other configurations from the parameters, otherwise a standard layer of “layerName” is added only. For the first layer, in both cases, 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.

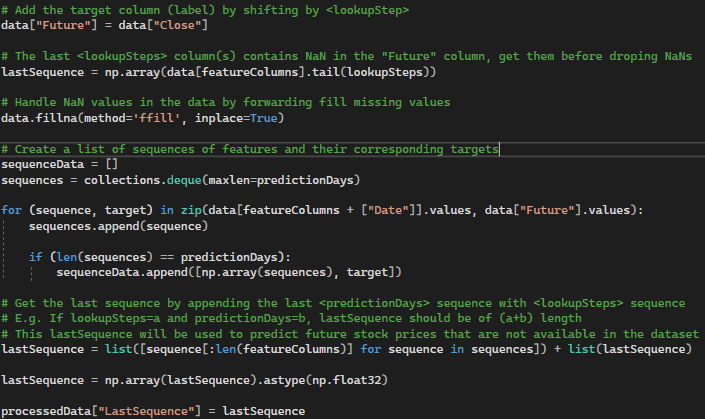


* “mean\_absolute\_error”: This is a common loss function used in regression problems, mostly the issues involving time series predicting. It measures the average absolute difference between the predicted values and the actual target values.
* “rmsprop”, stading for Root Mean Square Propagation, is an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

**TESTING**

In the phase of creating and training the model, I have modified the “dataProcessing.py” file:

* In the “processData()” method:

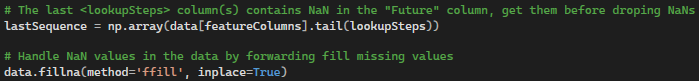




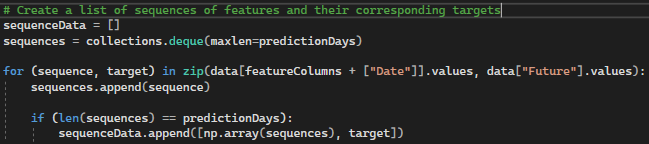
In this added code, a new column of “Future” is created in the dataframe, derived from the “Close” column by shifting by “lookupSteps”. It is the target for the Machine Learning predictions.



After shifting the "Close" values to create the "Future" column, the last “lookupSteps” rows will contain NaN values in the "Future" column because there's no actual future data available beyond the last data point. This "lastSequence" will be used later for predicting future stock prices that are not present in the dataset.

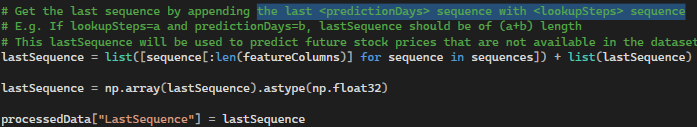


The next step is creating a list of sequences of features and their corresponding targets (“sequenceData”), each sequence having a length of "predictionDays":



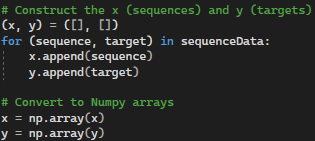
* An empty list "sequenceData" and a double-ended queue sequences with a maximum length of "predictionDays" are initialized. The purpose of sequences is to store a rolling window of sequences of features.
* A loop iterates through the rows of the data, extracting both sequences of features (specified by "featureColumns") and their corresponding target values (from the "Future" column). For each row, the feature sequence is appended to sequences.
* When the length of sequences reaches the specified "predictionDays", a new entry is added to "sequenceData". This entry contains the sequence of features (converted to a NumPy array) and the corresponding target value.

The method then constructs the “lastSequence” used for predicting process, by appending the last “predictionDays” sequence with “lookupSteps” sequence, before being converted to a NumPy array and cast to the float32 data type to ensure consistency in data types, and stored in the output “processedData” of the method.

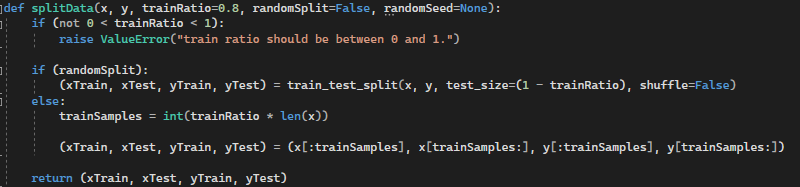


Two lists, “x” and “y” are then initialized, which are used to construct the input sequences (features) and their corresponding target values for training a machine learning model.

* The "sequenceData" list contains pairs of sequences and their corresponding target values. Each pair consists of a sequence of historical data points and the target value that needs to be predicted.
* The "for" loops through each pair ("sequence", "target") in "sequenceData".
* For each pair, the "sequence" (a sequence of historical data points) is appended to the "x" list, and the "target" (the corresponding target value) is appended to the "y" list.
* After this loop, "x" will contain all the input sequences, and "y" will contain all the corresponding target values. They will also be converted to Numpy arrays.



* In the “splitData()” method:



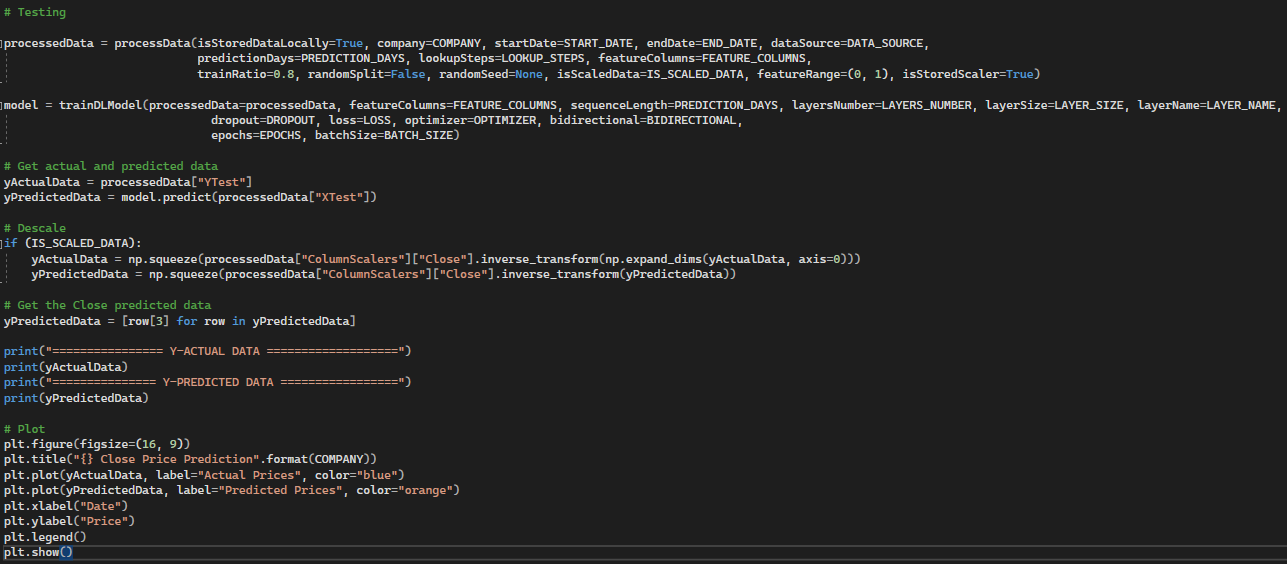
If “randomSplit” is “True”, the function uses the “train\_test\_split” function from “scikit-learn” to perform a random split of the data.

* “x” and “y” are split into training and testing sets. The “test\_size” parameter is calculated as (1 – “trainRatio”), which ensures that the specified ratio of data is used for training.
* The “shuffle” parameter is set to “False”, meaning the data will not be randomly shuffled before splitting.

Else, if “randomSplit” is “False”, a sequential split of the data is performed.

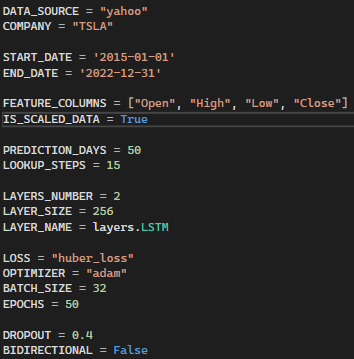
* The number of samples to be included in the training set (“trainSamples”) is calculated as an integer value of “trainRatio” multiplies the total number of samples (“x”) in the dataset.
* The data is split into training and testing sets based on the calculated “trainSamples”. The first “trainSamples” samples are assigned to the training set, and the remaining samples are assigned to the testing set, corresponding with “x” and “y” arrays.

The next phase is testing the model and plotting the data.

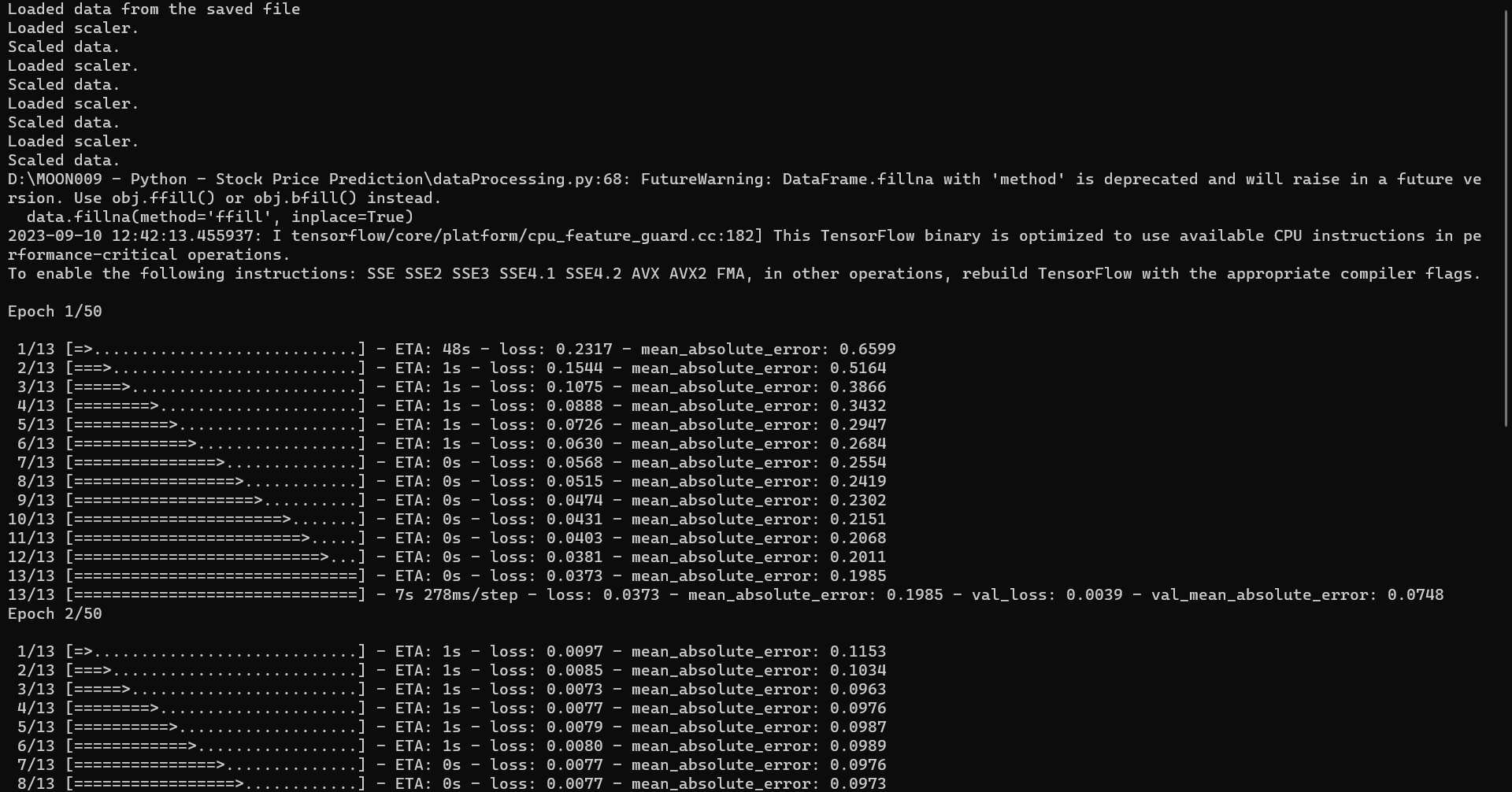


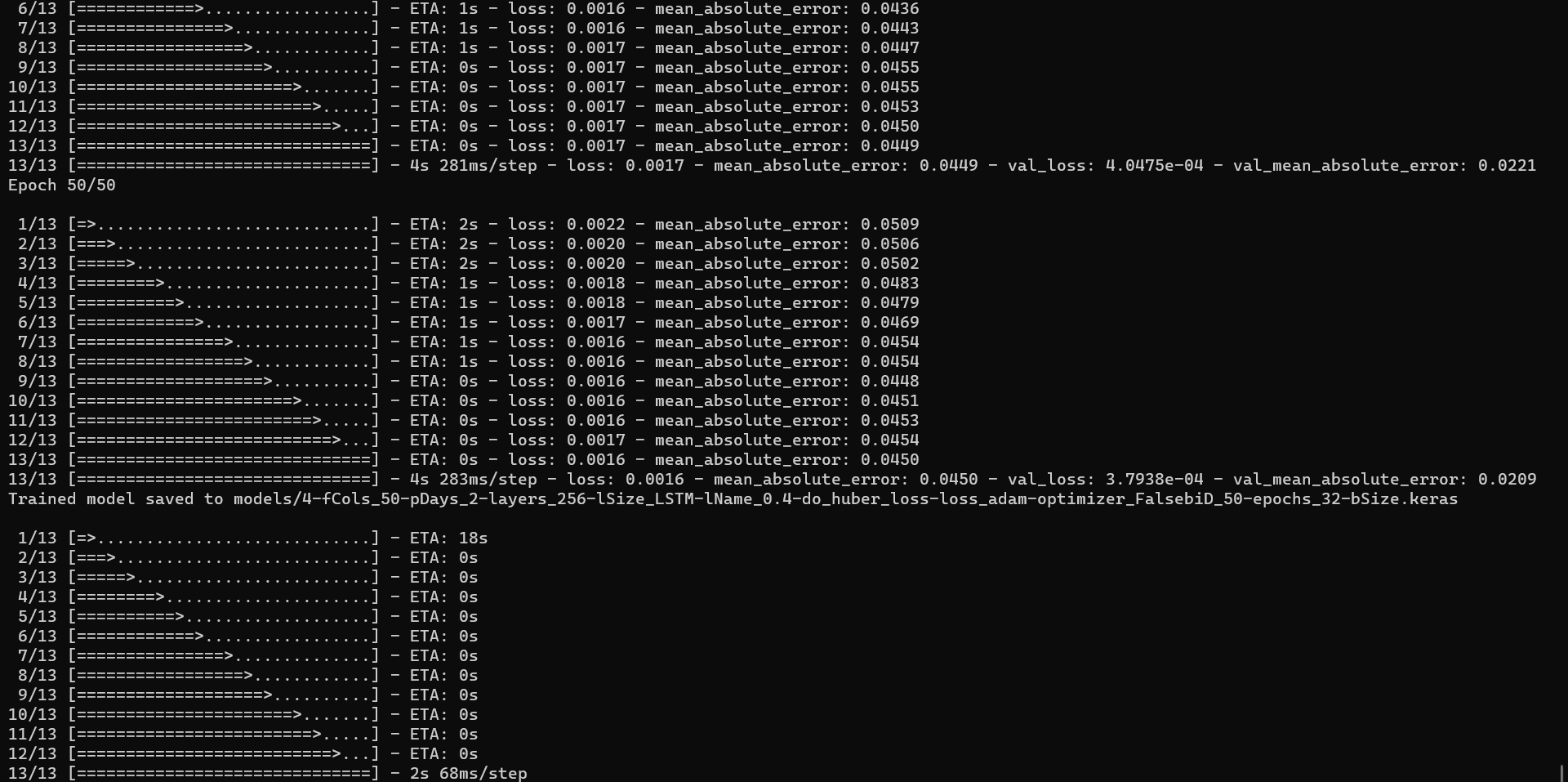
**1. Test with LSTM layer**

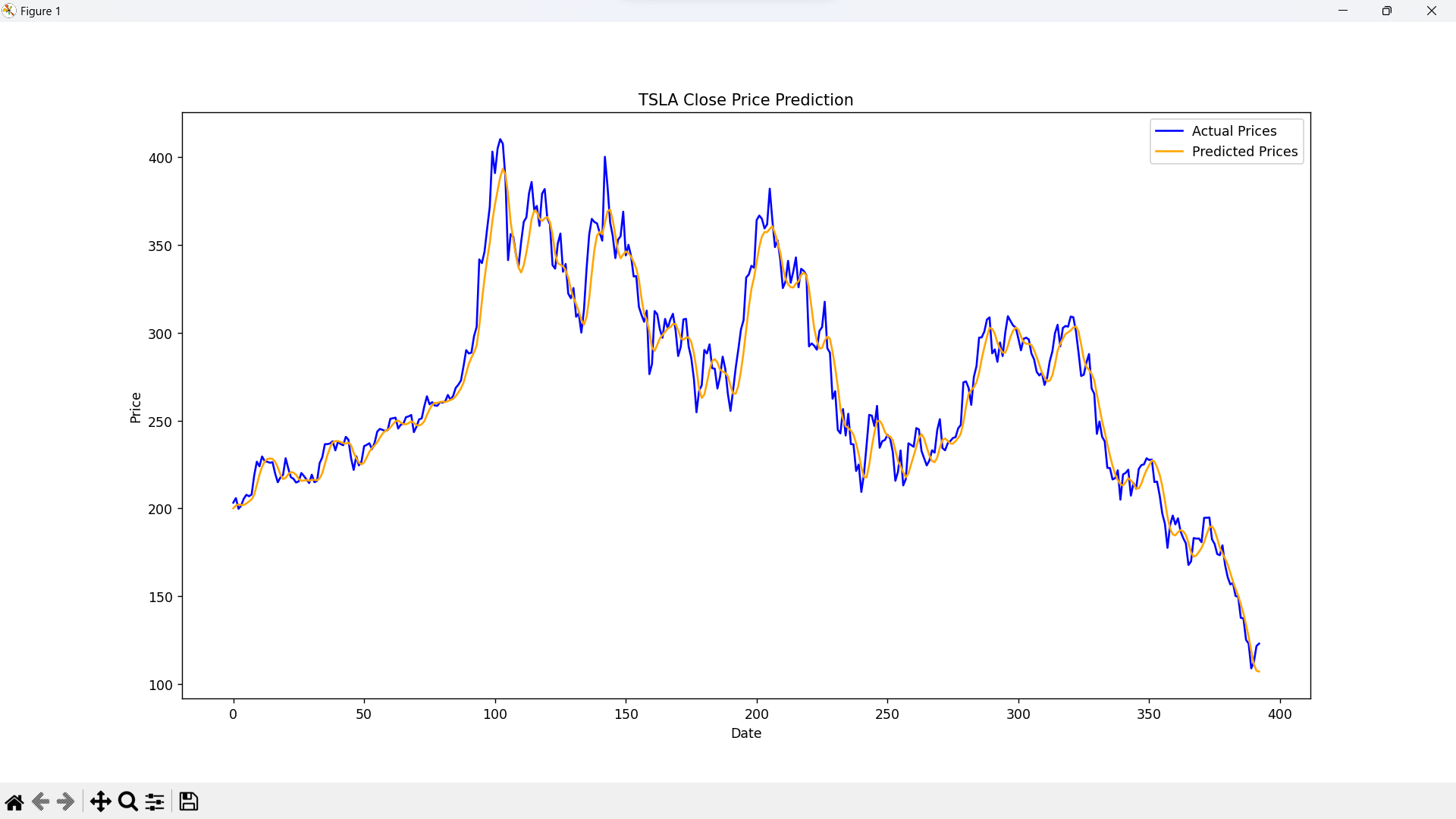
For the first test, some parameter I have used for this test:

**

Output:

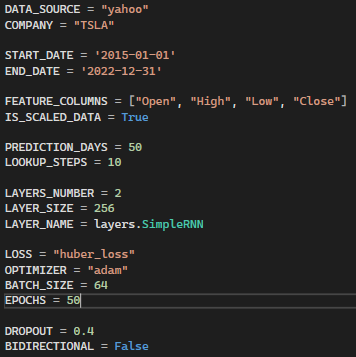




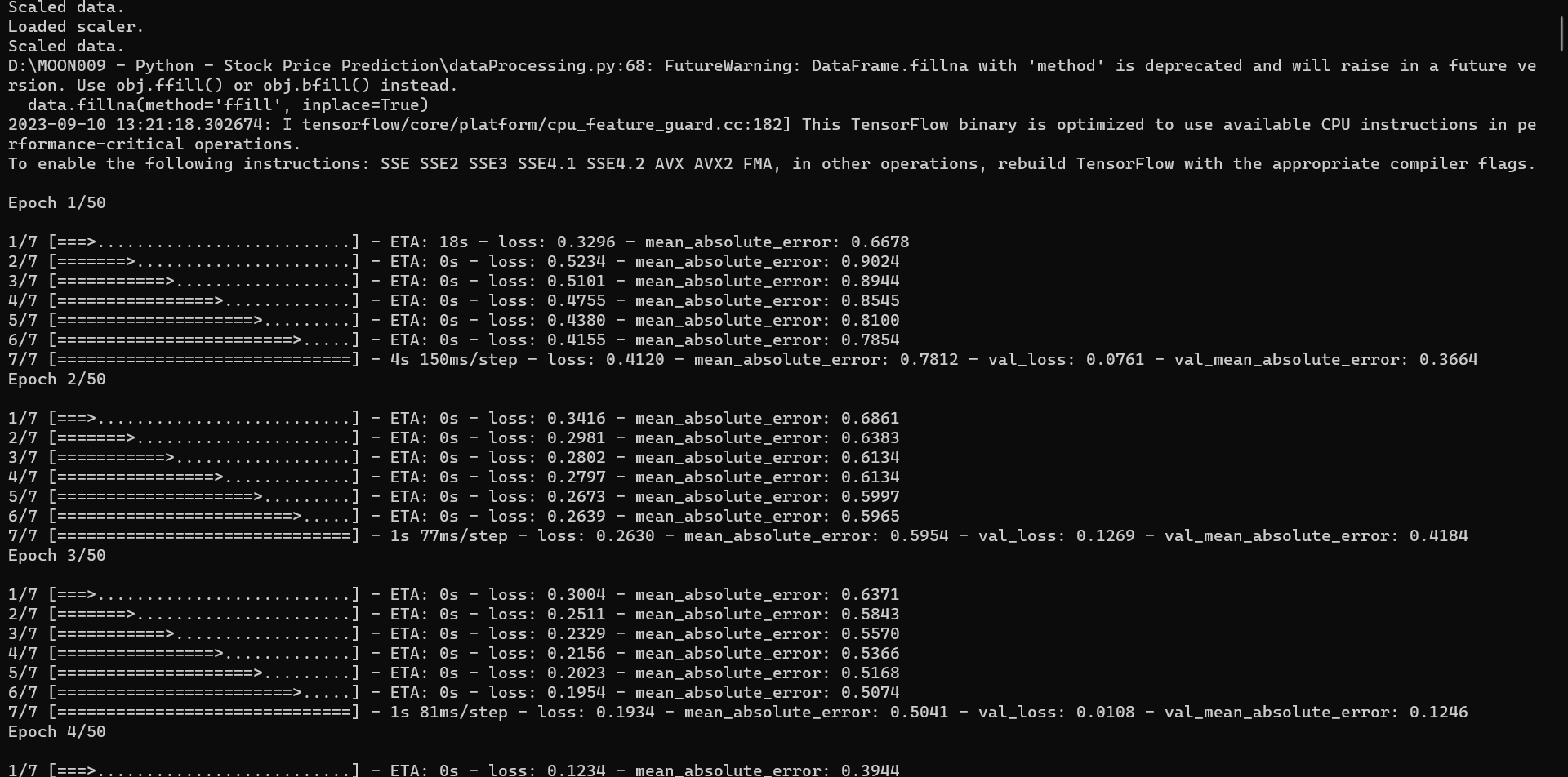


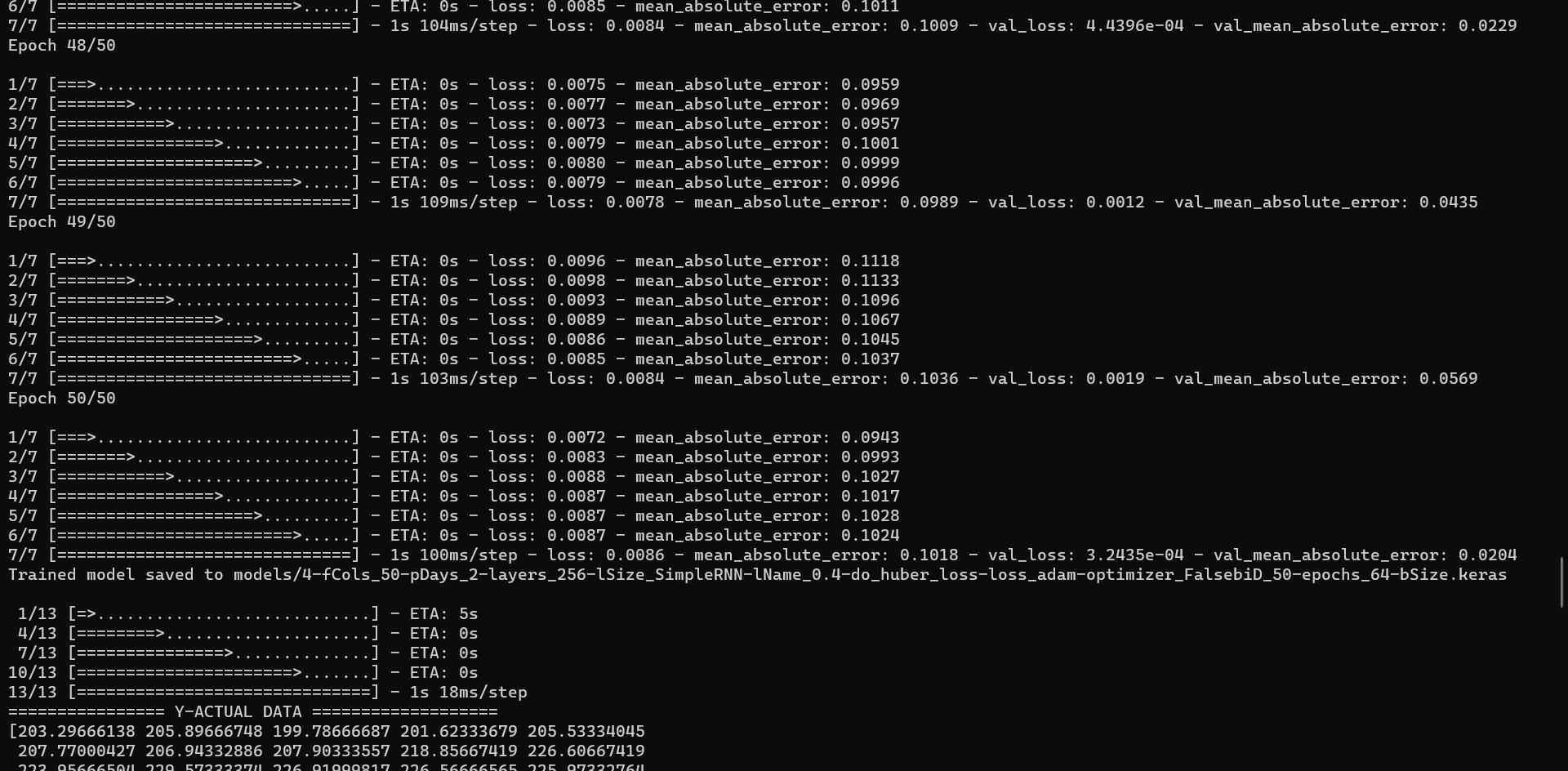
**2. Test with RNN layer**

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



Output:

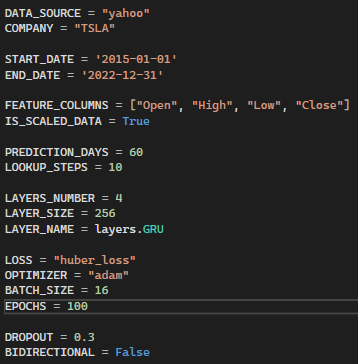






**3. Test with GRU layer**

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



Output:

