ASSIGNMENT-3
WEATHER TIME SERIES FORCASTING

SUMMARY

Siva Sai Madhumitha Kotala
KSU ID : 811257709

We will illustrate the main differences between timeseries data and the other dataset types we've already worked with using this temperature-forecasting exercise. We will see that recurrent neural networks (RNNs), a novel type of machine learning technique, totally flourish at addressing this kind of problem, whereas convolutional and highly linked networks are unable to handle this kind of dataset.

In all of our investigations, a total of 50% of the data will be used for training, 25% for validation, and the remaining 25% for testing. When working with timeseries data, it is essential to use validation and test data that is more recent than the training data because our objective is to predict the future based on the past rather than the opposite. This should be reflected in the validation/test splits. The issue will be precisely formulated as follows: Is it possible to calculate the temperature of a given day using data that was gathered every hour for the previous five days?

In order to train a neural network to use the data, let's begin by preprocessing it. Easier said than done—since the data is numerical in the first place, vectorization is not necessary.

In order to analyze time series data, we created 14 different models. Using common-sense techniques, the first model provided a baseline and produced a Mean Absolute Error (MAE) of 2.62. After that, we developed a simple machine learning model with a dense layer, which produced an MAE of 2.65 that was marginally higher. The flattening of the time series data, which eliminated the temporal context, resulted in poor performance of the dense layer model.

Additionally, a convolutional model was used, but it produced subpar results since it treated every data segment equally—even after pooling—disturbing the sequential order of the data.

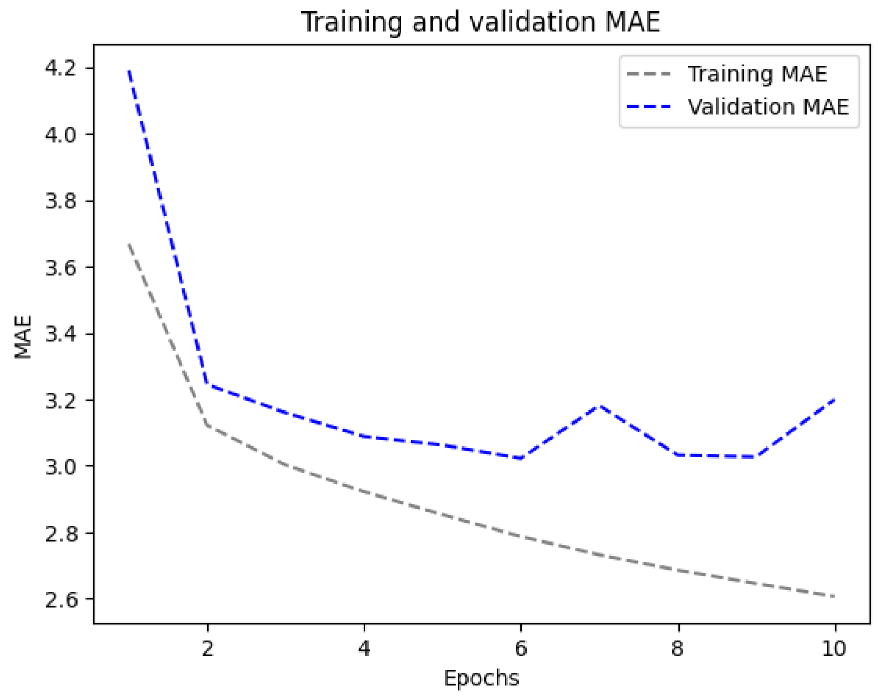
We concluded as a result that Recurrent Neural Networks (RNNs) perform better with time series data. Recurrent neural networks' (RNNs') ability to integrate knowledge from previous steps into current decision-making is a crucial component. The network can then use this to find patterns and dependencies in the sequential data. The RNN can represent sequences of different lengths because of its internal state, which functions as a kind of memory and retains data from previous inputs. But often, the fundamental Simple RNN is too straightforward to be very useful. The graphical representation indicates that Simple RNN consistently performs the worst out of all the models, which is a notable downside. Although the renowned "vanishing gradient problem" causes Simple RNN to struggle operationally, particularly in deep networks, it should be able to retain knowledge from all prior time steps theoretically. The network is essentially untrainable because to this issue. More sophisticated RNN variations were created in response to this difficulty and are incorporated into Keras as the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM). The simplest GRU model produced the greatest results from our experiments out of all the models, mostly because it is more computationally efficient than LSTMs and can capture long-range dependencies in sequential data.

We tested six different LSTM models with different units in stacking recurrent layers (8, 16, and 32), and the model with 8 units showed the greatest performance. LSTMs are a well-known architecture for handling time series data successfully. We also experimented with bidirectional data presentation to improve accuracy and solve the forgetting problem, and we used recurrent dropout to prevent overfitting. Similar MAE values, which were consistently lower than the common-sense model, were displayed by all of these LSTM models.

Finally, we tried to integrate an RNN with a 1D convolution model. The hybrid model produced a greater mean absolute error (MAE) of 3.95, which can be attributed to the limits of the convolution in preserving the information order. My findings suggest that basic RNNs should be avoided for time series analysis since they have trouble with the vanishing gradient issue and are unable to accurately capture long-term relationships. Instead, take into account more sophisticated

RNN architectures that are intended to get around these obstacles, including LSTM and GRU. Although GRU may provide more effective outcomes than LSTM, our trials indicate that LSTM is a popular option for processing time series data. Hyperparameters like the number of units in stacked recurrent layers, recurrent dropout rates, and the usage of bidirectional data presentation can all be tuned to improve GRU models. Additionally, since the combination of RNN with 1D convolution did not produce the best results, it is advised to concentrate on RNN designs designed for sequential data. Convolutional methods are less appropriate for time series data processing since they frequently cause information to be out of order.

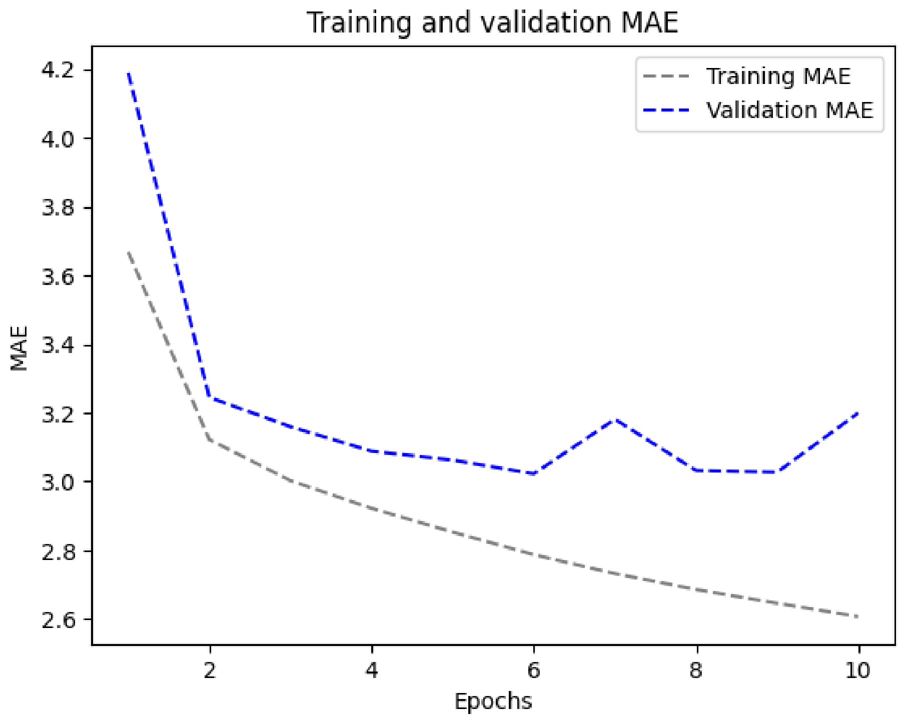
A basic machine-learning model:



1D convolutional model:

A convolutional model would be adequate in terms of using appropriate architectural priors, considering that the input sequences are made up of daily cycles. Just as a spatial convolutional network has the potential to reuse representations in several locations within an image, a temporal convolutional network may also employ the same representations throughout various days.

Because not all meteorological data satisfies the translation invariance condition, this model performs significantly worse than the densely linked model; it only accomplishes a validation MAE of about 3.1 degrees, which is far from the reasonable baseline.



A Simple RNN:

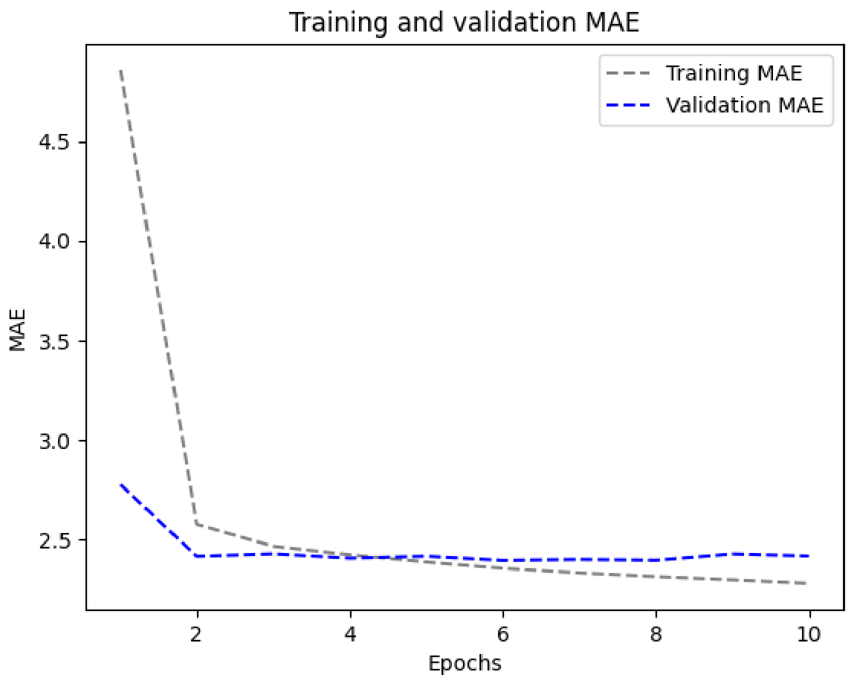
Recurrent neural networks (RNNs) are able to recognize intricate relationships and patterns in sequential data because of their remarkable ability to integrate prior time step information into current decision-making processes. Sequences of varying lengths can be described since the internal state of an RNN serves as a memory for prior inputs. Although a basic RNN may theoretically maintain data from all previous times, practical challenges occur. This leads to the vanishing gradient problem, which makes training deep networks difficult. Additionally, the graph shows that out of all of them, the simplest RNN performs the worst.

To overcome this problem, as a part of Keras, we have to create LSTM and GRU RNNs.

A Simple GRU (Gated Recurrent Unit):

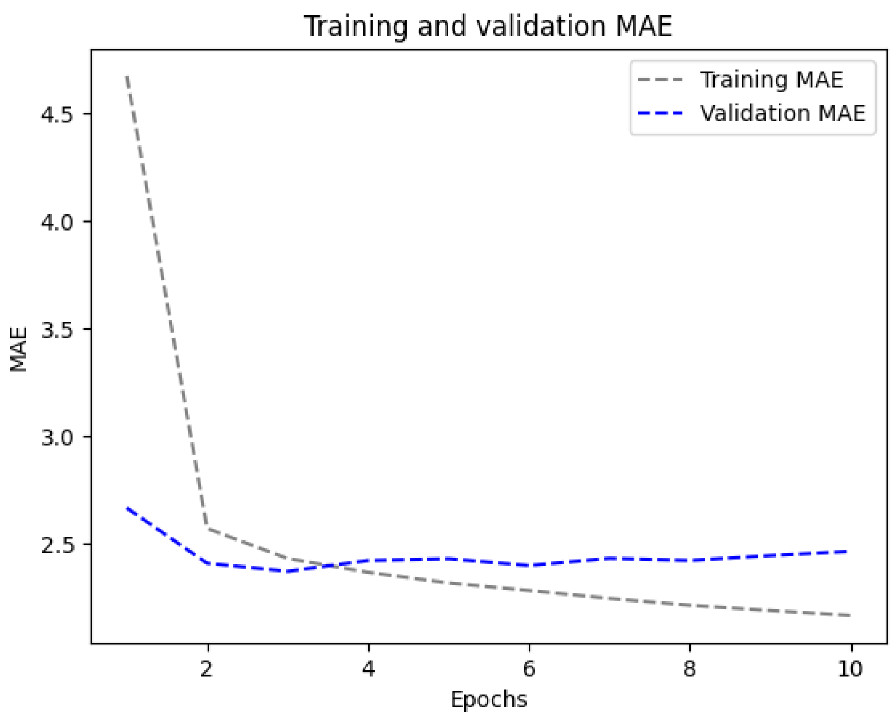
We will use Gated Recurrent Unit (GRU) layers in place of LSTM layers. GRU and LSTM are somewhat similar; think of it as a simplified, more basic version of the LSTM architecture.

In comparison to the other models, we found that Test MAE is 2.57 is the most efficient model. It also requires less computing power than Long Short-Term Memory (LSTM) models and accurately captures long-range dependencies in sequential data.



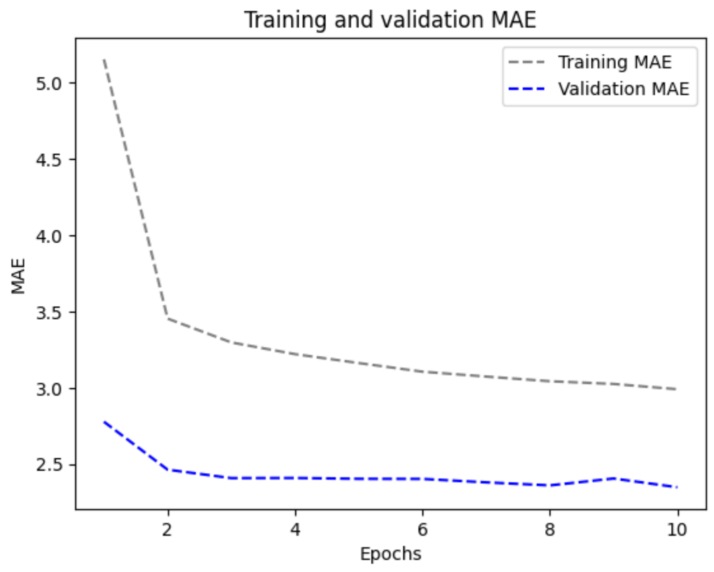
LSTM-Simple:

For this specific use case, recurrent neural networks offer a class of neural network topologies. A highly favored layer among them is the Long Short-Term Memory (LSTM) layer. In a minute, we will see how these models perform by first testing the LSTM layer.



Much better! We find that the validation MAE is as low as 2.46 degrees, while the test MAE is 2.53 degrees. Finally, the LSTM-based model shows how effective machine learning is in this quest by outperforming the common-sense baseline (although only considerably, at least for the time being).

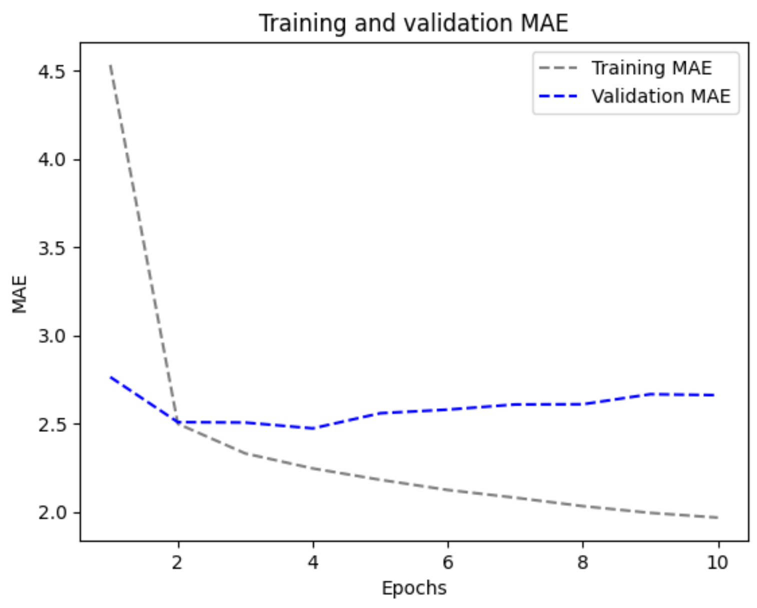
LSTM - dropout Regularization:



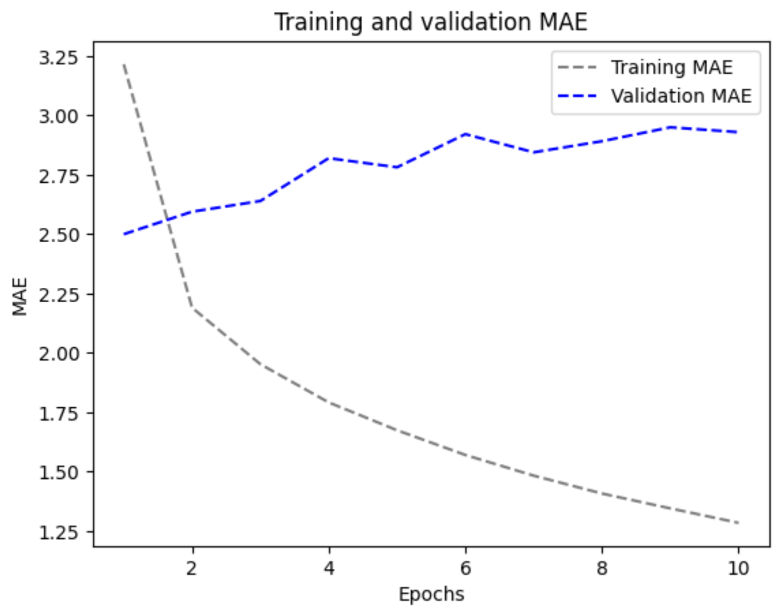
Success! Overfitting, which was present for the first five epochs, has stopped. We achieve a validation MAE as low as 2.34 degrees and a test MAE of 2.61 degrees. Nothing too bad.

I created six different LSTM models with 8, 16, and 32 units as the different numbers of units inside the stacked recurrent layers.

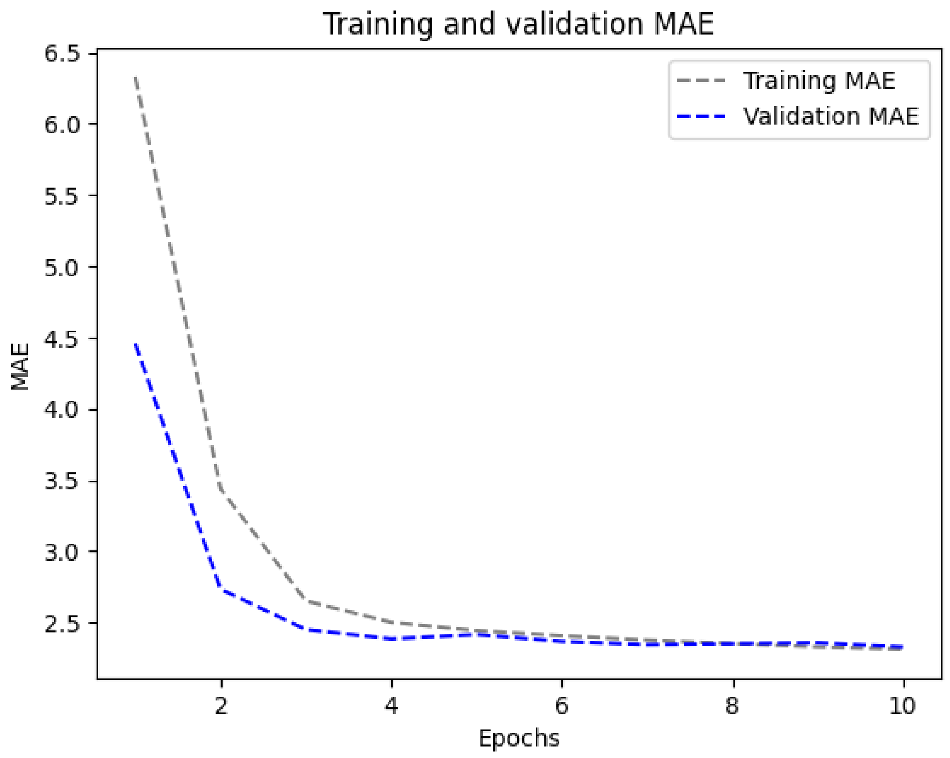
Stacked setup with 16 units



Stacked setup with 32 units:



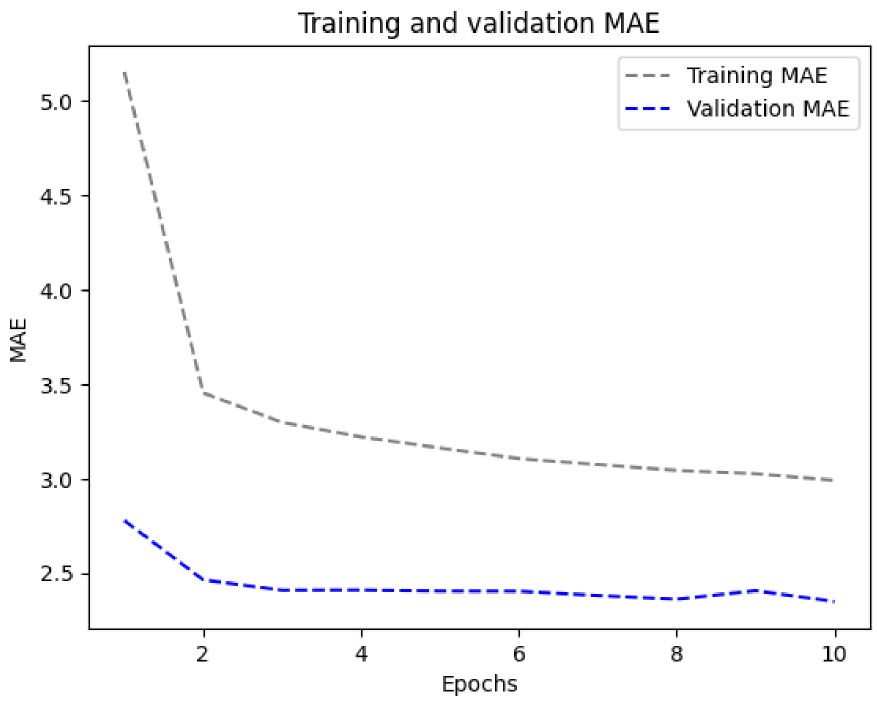
Stacked setup with 8 units



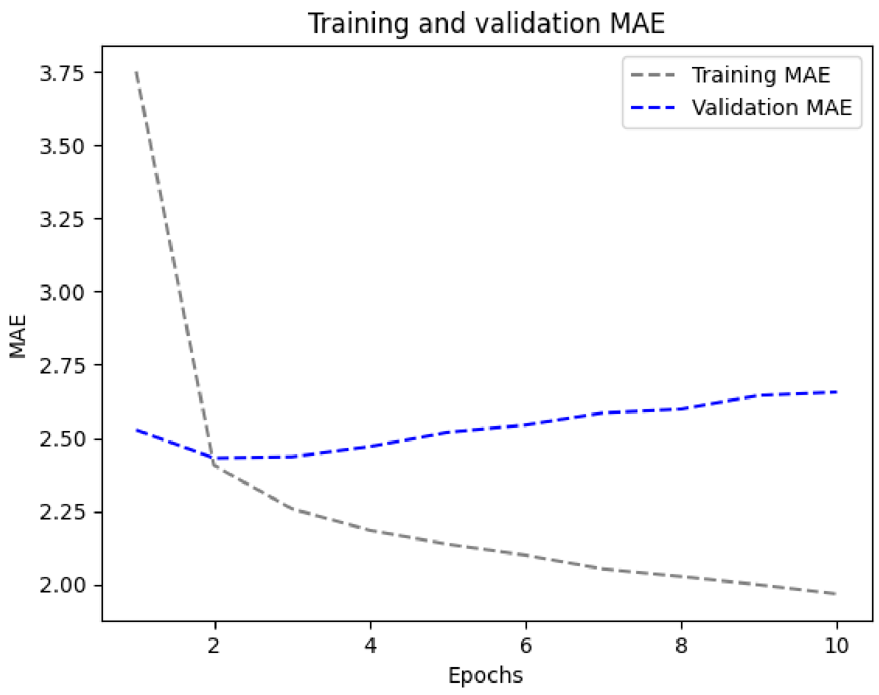
Among these options, the 8-unit arrangement performed the best, with an MAE score of 2.49.

In order to improve the model, I also experimented with techniques like bidirectional data, which decreases MAE values by providing information to a recurrent network in a variety of ways, and recurrent dropout, which counteracts overfitting. All of the models' MAE values were comparable as a result of these modifications, and most amazingly, they were consistently lower than those of the common-sense model, as the MAE assessment graph clearly demonstrates.

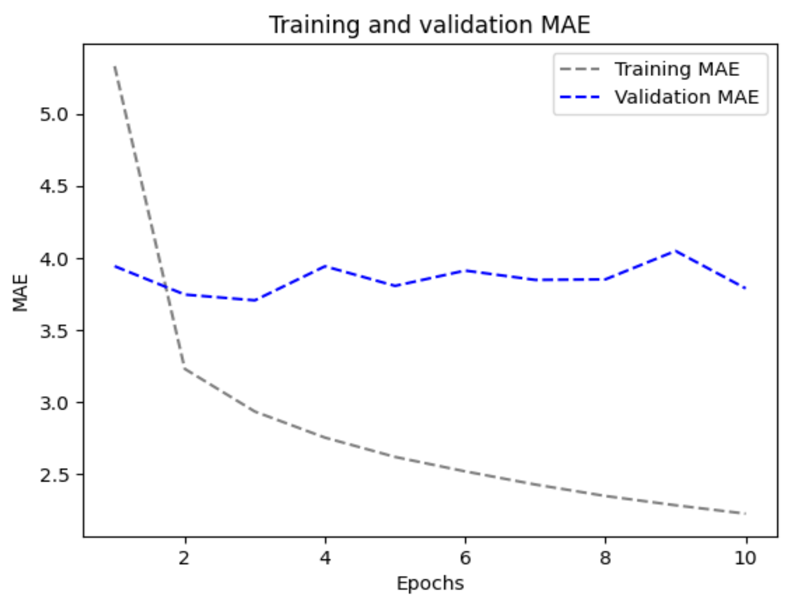
LSTM - dropout-regularized, stacked model:



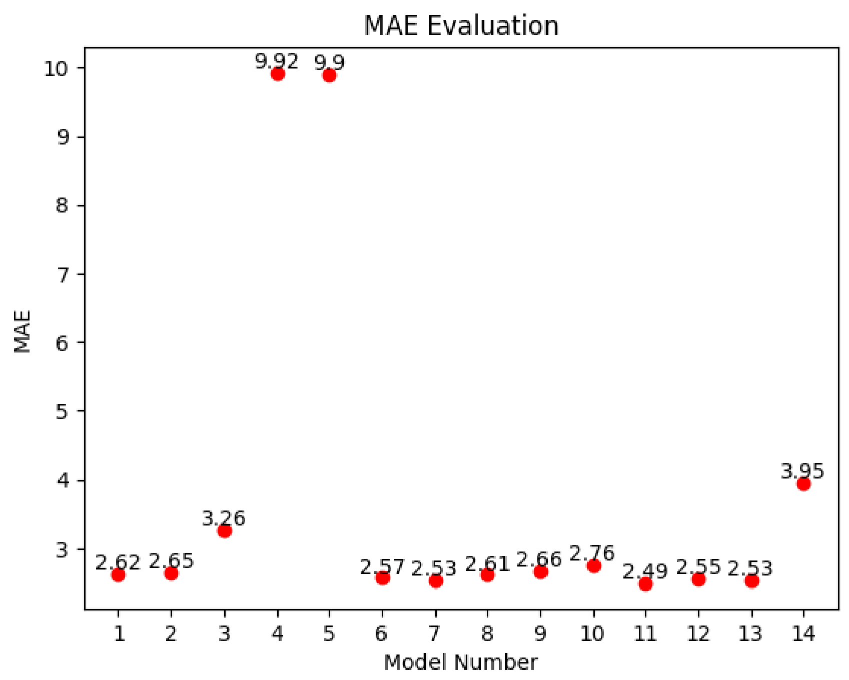
Bidirectional LSTM:

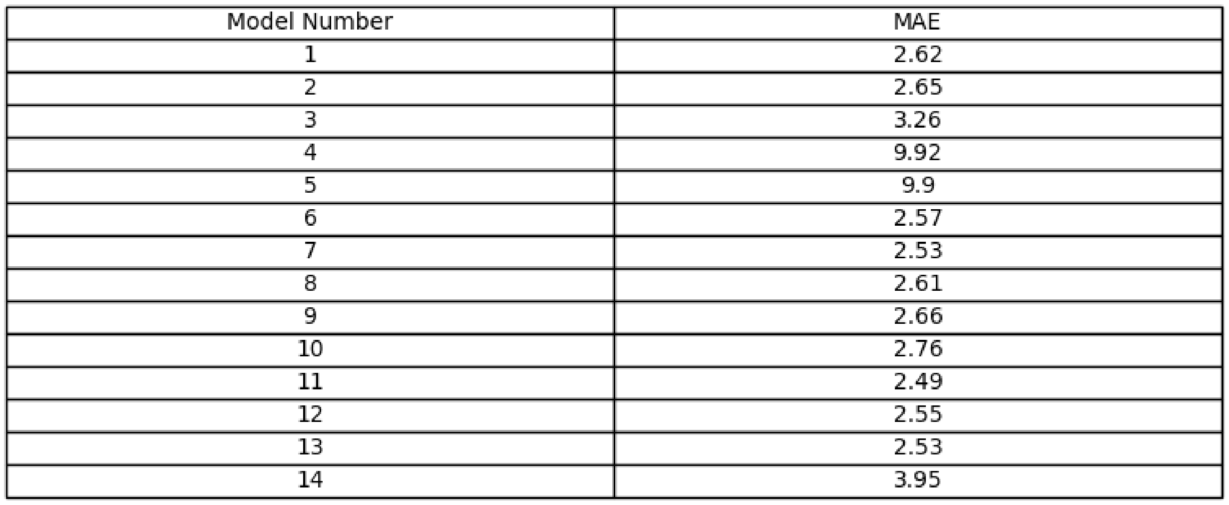


1D Convnets and LSTM together:
With 3.95 MAE, the model I created with RNN and 1D convolution yielded insufficient results. This poor performance could be the result of the convolution limit destroying the information order.



MAE Evaluation:





Model Number	MAE
1	2.62
2	2.65
3	3.26
4	9.92
5	9.9
6	2.57
7	2.53
8	2.61
9	2.66
10	2.76
11	2.49
12	2.55
13	2.53
14	3.95

In conclusion, my findings indicate that using LSTM and GRU (advanced RNN architectures) is the preferable choice, while combining RNN with 1D convolution yielded subpar outcomes. Although bidirectional LSTM is still a popular alternative, I think that GRU is a more efficient method for processing time series data after considerable testing. The number of units in the stacked recurrent layers, the recurrent dropout rate, and the utilization of bidirectional data are examples of hyperparameters that should be changed to optimize GRU.