Assignment 3: Time-Series Data

Kavya nagini madhu lasya Madineni

The main aim of this assignment was to use Recurrent Neural Networks (RNNs) to analyze time-series data for weather forecasting. More specifically, the task was to predict the temperature a day ahead by looking at a 5-day history of weather measurements. Throughout the assignment, we looked into various strategies to enhance the network's performance and experimented with different deep-learning layers tailored for time-series data. The methods we implemented included tweaking the number of units in the recurrent layers, comparing LSTM and GRU architectures, and even exploring hybrid models like CNN-LSTM.

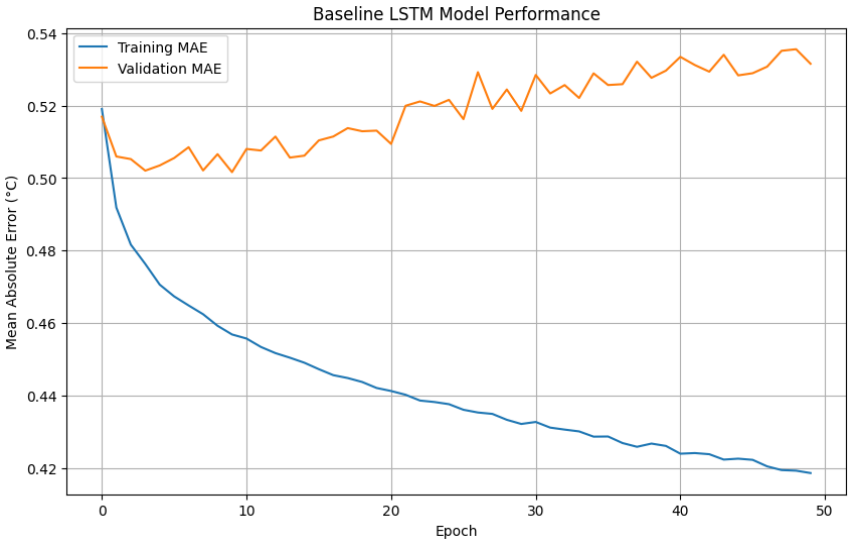
DataSet:

For this assignment, we utilized the Jena Weather Dataset, which includes weather data collected every 10 minutes from 2009 to 2016. We took some time to preprocess the dataset by normalizing the features and setting up time-series windows of 120 timesteps, which corresponds to a span of 5 days. Our main focus was on predicting the temperature, looking ahead 24 hours into the future.

Implemented Models:

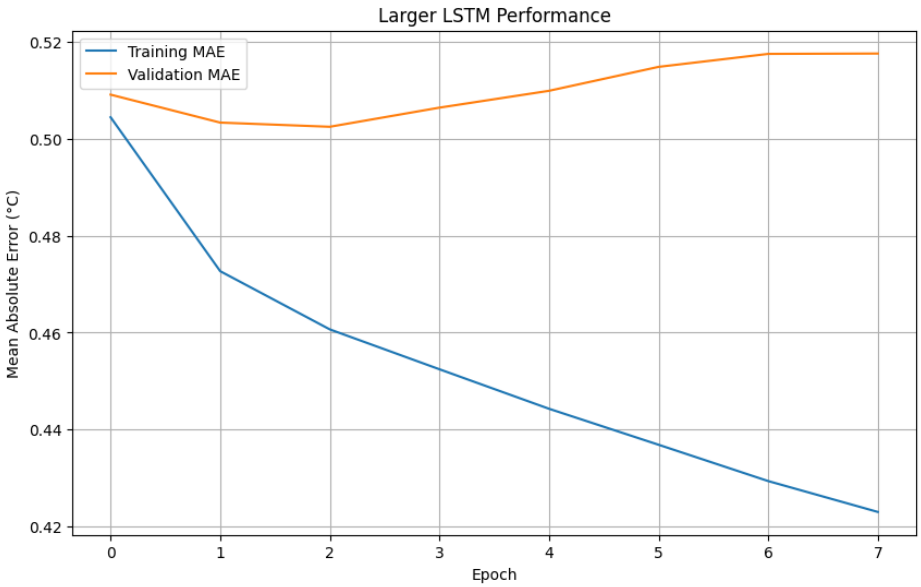
Baseline LSTM

The baseline model was built with just one LSTM layer that had 32 units, along with a recurrent dropout of 0.25 and a standard dropout of 0.5. This setup gave us a solid foundation for comparison, resulting in a test Mean Absolute Error (MAE) of 0.4572. However, we noticed some overfitting after the 10th epoch, as the validation MAE started to rise while the training MAE kept on dropping.



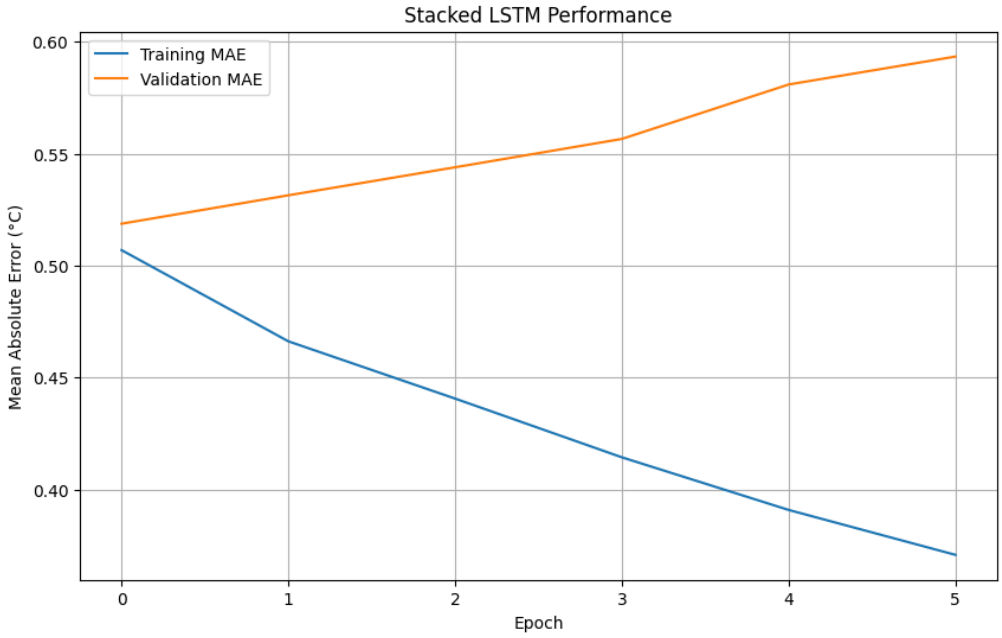
Larger LSTM (64 Units)

To boost performance, we decided to enhance the baseline model by doubling the number of LSTM units to 64. This adjustment led to a test MAE of 0.4492, which is a slight improvement of 1.75% compared to the baseline. However, we still faced issues with overfitting, which showed up earlier in the training process than it did with the baseline model.



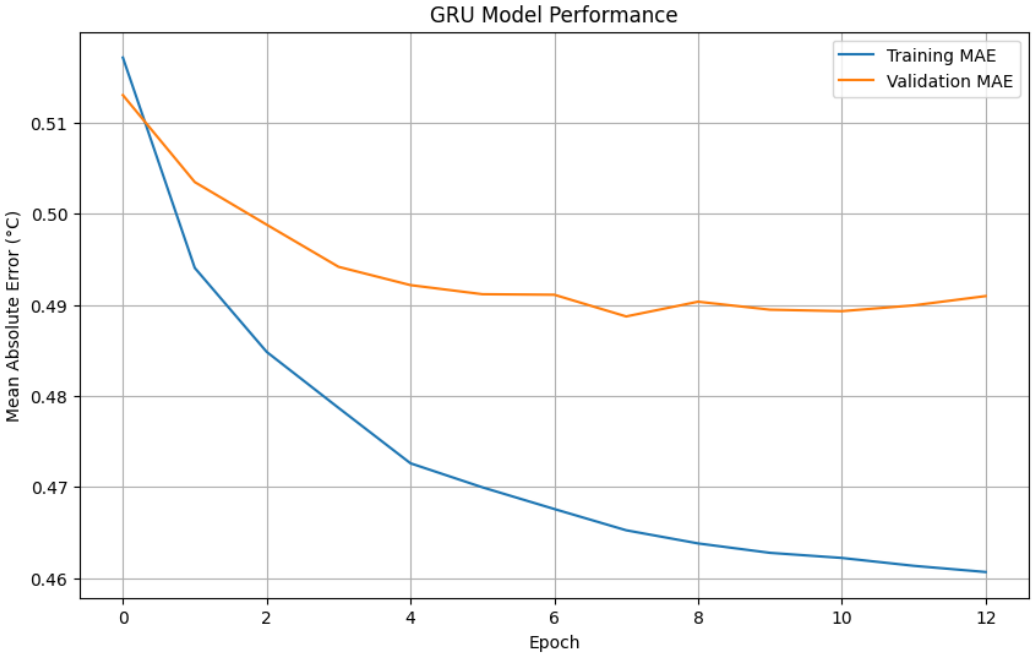
Stacked LSTM (64 → 32 Units)

We set up a stacked architecture to effectively capture hierarchical features by utilizing two LSTM layers—one with 64 units and the other with 32 units. However, despite its theoretical potential, this model didn't quite hit the mark compared to the baseline, ending up with a test MAE of 0.4678, which is 2.31% worse than the baseline. It seems that the added complexity may have led to some overfitting.



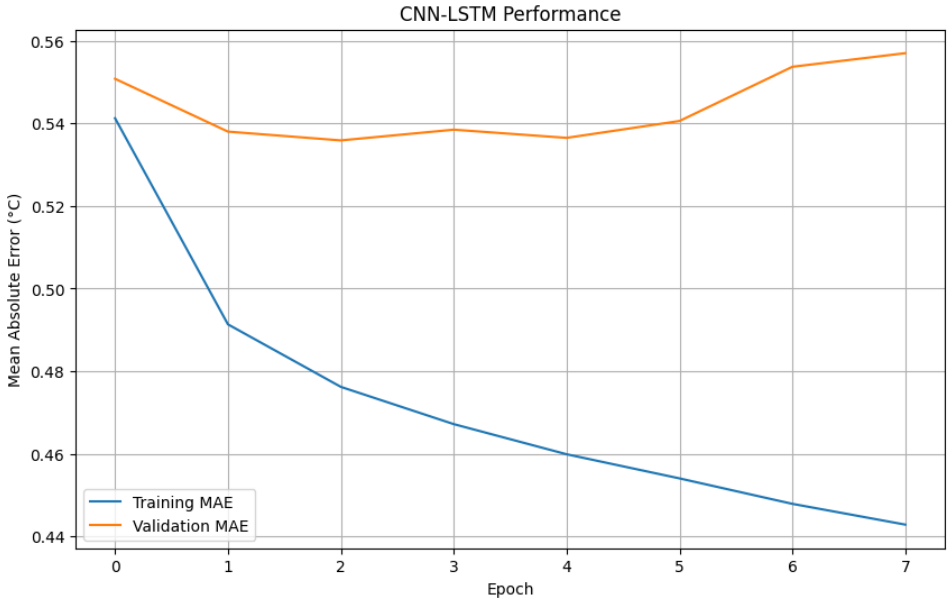
GRU Model

The GRU model swapped out LSTM layers for GRU cells, which come with a more straightforward gating mechanism and are easier on computational resources. This model hit a new high with a test MAE of 0.4389, marking a notable improvement of 4% over the baseline. Plus, GRUs showed quicker convergence and outperformed LSTMs in terms of generalization.



CNN-LSTM Hybrid

We put together a hybrid architecture that uses convolutional layers to pull out spatial features, paired with an LSTM layer to handle temporal modeling. While it showed promise in picking up local patterns, this model had a tough time managing the long-term dependencies that come with weather data. In the end, it ended up with a test MAE of 0.4930, which made it the least effective model in our tests.



Summary of Results

Model	Test MAE	Improvement Over Baseline	Key Observations
Baseline LSTM	0.4572	___	Strong baseline; overfitting epoch ~10
Larger LSTM	0.4492	-1.75%	Modest improvement; overfitting after epoch ~8
Stacked LSTM	0.4678	+2.31%	Increased complexity led to worse performance
GRU Model	0.4389	-4%	Best performer; computationally efficient
CNN-LSTM	0.4930	+7.83%	Struggled with long-term dependencies

Key Insights

- The GRU model really stood out, achieving a test MAE of 0.4389, which showcases just how efficient and effective it is for this task.
- When we increased the model's capacity with a larger LSTM, we saw only slight improvements, but it still faced issues with overfitting.
- Adding stacked architectures made things more complex without delivering any significant performance boosts.
- Hybrid architectures that mixed convolutional layers with RNNs had a tough time, mainly because they struggled to capture long-term dependencies.
- Overfitting was a common issue across all models, highlighting just how crucial regularization techniques like dropout are.