The Next Wave of Meteorology: An Evaluation of Deep Learning Models for Weather Forecasting

Vaibhav Malhotra
Georgia Institute of Technology
vaibhavmalhotra99@gatech.edu

Pranav Sharma
Georgia Institute of Technology
psharma373@gatech.edu

Koushik Karan Geetha Nagaraj
Georgia Institute of Technology
knagaraj31@gatech.edu

Shashank Garikipati
Georgia Institute of Technology
sgarikipati7@gatech.edu

Abstract

Weather forecasting plays a vital role in deriving insights across various climatic predictions. Numerous networks have been implemented to focus on this domain, producing forecasting results in an autoregressive manner. In our study, we compare and perform analysis on various models such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, Transformers, Temporal Convolutional Networks (TCN), and our proposed modified TCN to uncover the spatio-temporal relationships within weather data. Specifically, we predict all 21 features of the dataset but focus our analysis on 6 key features: air pressure, temperature, relative humidity, air density, CO2 concentration, and precipitation. Our results and comparisons have shown that the modified TCN network, coupled with a modified loss function, yields the best forecasting results for these six features using the Jena Weather Dataset. Our rigorous analysis can be viewed in the Results and Appendix section. Our Implementation can be found on GitHub.

1. Introduction and Motivation

We tried to solve the problem of weather prediction. We explored several models with varying assumptions and complexities including architectures such as Recurrent Neural Networks (RNN), Long-Short Term Memory (LSTM), Transformers, Temporal Convolutional Networks (TCN) and modified a Temporal Convolutional Networks (TCN). The goal of this project is to identify which of these approaches gives the best predictions primarily for the following variables: air pressure, temperature, relative humidity, air density, Co2 concentration, and precipitation.

Today, weather prediction primarily relies on computer simulation, accounting for numerous factors such as temperature, humidity, wind speed, and atmosphere pressure, different weather stations, etc. Currently, there are two tools that form the foundation to weather forecasting: Numerical Weather Prediction (NWP) and Ensemble forecasting. Numerical Weather Prediction as discussed in [1] computes weather variables by partitioning the atmosphere into three parts (Hemisphere Synoptic, and Mesosphere) and simulating thermodynamic processes such as convection, radiation, and atmosphere-earth interaction. Ensemble forecasting as introduced in [2] is used in tandem with NWP models. Several NWP models are initialized with slight variations in initial conditions (to account for uncertainty) to provide probabilistic forecasts [3]. There are several challenges associated with the current approaches. Firstly, small-scale weather patterns are difficult to capture due to limited spatial resolution. Secondly, atmospheric processes are highly non-linear and complex, meaning modeling and simulating all processes is difficult. Finally, Extreme weather conditions are difficult to predict due to high non-linearity.

The current research [4] on this involves several implementations of deep learning models such as ConvLSTM and Stochastic Adversarial Video Prediction (SAVP) to forecast temperature. Although deep learning models can't yet match the accuracy of the most advanced NWP models, this work[4] shows that they can be used to accurately extend the forecast range beyond nowcasting. Their potential could be further increased by additional developments in neural network topologies and training techniques, which would make them an attractive addition to or substitute for conventional weather forecasting models. This study highlights the potential of advanced deep neural networks to improve weather forecasting and possibly reduce resource requirements. We mainly aim to predict the six features above mentioned.

The dynamic nature of weather and climate significantly impacts various sectors such as agriculture, travel, and con-

struction. Accurate weather and climate forecasting is crucial for effective planning and strategy in these areas. However, the variability in weather patterns and the limitations of existing models make it difficult to consistently provide reliable forecasts. Developing a model that can accurately forecast weather conditions reliably is therefore essential.

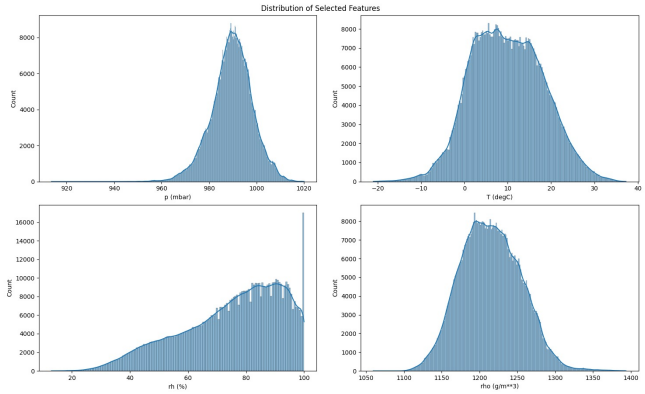

Figure.1. Features Distribution

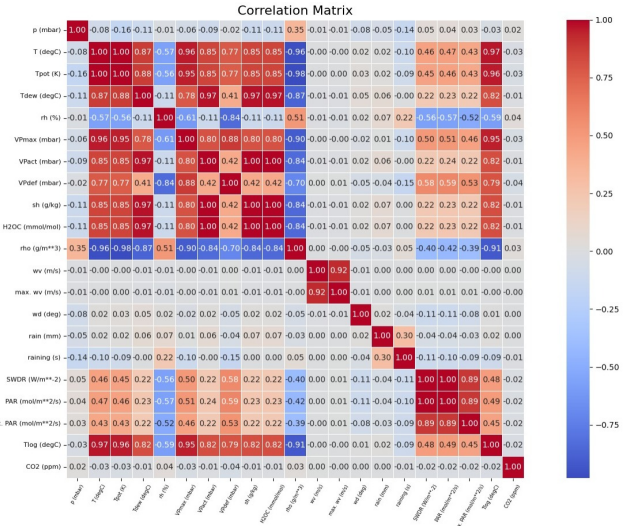

Figure.2. Features Correlation Matrix

We utilized the Jena dataset for this project. The Jena Dataset consists of 21 variables recorded at the Jena weather station in Germany. (A full list of variables is available in the appendix). Each Measurement (data point) is taken at 10-minute intervals and recorded from 2009 to present. While each of the models explored in this study predicts all 21 variables, the focus is set on 6 variables. are the following: Air pressure, Temperature, Relative Humidity, Air density, CO2 concentration, and precipitation. In Figure 1 the features distribution is plotted that shows the variance that is beneficial for our modeling.

The following 6 features are chosen based on their correlation matrix in Figure 2. The correlation matrix [5] used for machine learning feature selection is important as we want a minimum correlation between their features. Minimum correlation is meant by lower similar data that is desired for our context while forecasting.

2. Approach

During this project, four modern deep architectures for time series forecasting were evaluated: RNNs, LSTMs, transformers, and TCNs. The main idea of the project is to autoregressively predict the weather for a certain number of time steps. i.e. we aim to evaluate the model for 3,6, and 9 hours into the future. The hypothesis is that given enough data transformers and TCNs should outperform LSTMs and RNNs. RNNs[6] are expected to perform the worst among the architectures. However, due to limitations in computational resources and time, the simpler models can potentially outperform the complex ones.

2.1. Recurrent Neural Networks

In RNNs, temporal data is propagated using a hidden state. As sequential data is passed in, the hidden state is updated at each time step to carry information forward during prediction. The main benefit of Recurrent Neural Networks is their simple and straightforward architecture. Additionally, no position embedding is required since the approach in which the data is passed into the network is inherently step-by-step. However, they have several problems associated with training and inference. Firstly, They suffer from vanishing gradients and exploding gradients when dealing with large sequences. Additionally, they have a short memory, meaning during inference prediction becomes poor in comparison to the LSTMs, Transformers, and TCNs.

The process of preparing data for prediction begins by loading it from a CSV file using the pandas' library, ensuring that specific encoding is applied to handle character formats correctly. To maintain the integrity of the data, any missing values are dropped. Additionally, the 'Date Time' column, which is not utilized in the model, is removed. Finally, the data is scaled using Z-score normalization. The dataset is split into training and testing sets using an 80-20 split.

RNN is the baseline model as it was one of the first models with a spatio-temporal configuration. It retains memory to a certain level that aids it in remembering patterns and that in turn helps it to predict the future. This core concept will be used to forecast weather data for all the features. The RNN[6] is a simple model and it has several issues such as vanishing gradients. Although, it has several issues we still wanted to understand if this network would be sufficient for our data. Also to create a comparative baseline for other models. The main other advantage of an RNN is its simple implementation and its ability to reach convergence at a faster rate.

The model is compiled with the Adam optimizer and mean squared error loss function, which are standard choices for regression problems. Validation during training is performed using a subset of the training data to provide an unbiased evaluation of the model.

2.2. Long-Short Term Memory

LSTMs [7] is a specialized RNN designed to mitigate the issue of vanishing gradient and exploding gradient. They are significantly better at captures long-range dependencies and used ubiquitously for tie series prediction. All of the advantages of RNNs apply to LSTMs. However, due to the architecture being more complex, they are more computationally intensive. Additionally, Both RNNs and LSTMs process data sequentially, which causes slow processing of data.

Before training the model, the dataset underwent preprocessing to ensure that it's suitable for training. This included removing any missing values and selecting relevant features and target variables, wherein the detailed process has been described earlier. Additionally, the data is normalized using the StandardScaler to ensure that all features have a mean of 0 and a standard deviation of 1. This also helps in stabilizing the training process and allows the model to converge faster.

In the proposed architecture of the LSTM, the model is designed to capture the temporal dependencies present in the Jena dataset. The architecture begins with an input layer that accepts the preprocessed features from the dataset, followed by three LSTM layers in total. These LSTM layers handle sequences of data and retain long-term dependencies. The first LSTM layer has 128 units of neurons and subsequent layers have 64 units each. The ReLU (Rectified Linear Unit) activation function is employed in these layers to introduce non-linearity and enable the model to learn complex relationships within the data. Additionally, dropout layers have been added after the first and third LSTM layers to prevent overfitting by randomly dropping a fraction of input units during training, thus enhancing the model's generalization capability. Finally, following the LSTM layers, dense layers have also been incorporated to process the features extracted by the LSTM layers. Batch normalization layers have also been added within the dense layers to stabilize the training process by normalizing the activations. The model's output layer, with a linear activation function, predicts future values based on the learned patterns. Throughout training, the Adam optimizer minimizes the Mean Squared Error (MSE) loss.

2.3. Transformers

Transformers introduced the self-attention mechanism, in which features with higher dependence were given higher importance. This is highly intuitive for language modeling purposes. In the context of modeling weather, the days closer to the current days will most likely given more importance. Transformers significantly outperform LSTMs at handling long-range dependencies. Additionally, since data is no longer fully processed sequentially, training is highly parallelizable and, therefore, faster. However, since transformers have no inductive bias, they require an extremely large corpus of data to learn and predict reliably. They are computationally expensive to train and are highly resource-intensive.

Since transformers do not recognize the order of the data inputted in any manner, positional embedding is necessary to inform the model about the time step. Since weather involves cyclical data, the embedding for this use case involves a transformation of the month and time of day into an embedding feature. this feature is essentially the sum of the transformed time of day in minutes (12:00 pm would be 720) and the transformed month of the year (December would be 0.12, January would be 0.01). For example, the time at 11:00 am on 1/10/2024 would be 660.10. Year and day are ignored since only days and months are cyclic and provide value in terms of position embedding for transformers. When inputting to the model, the features and transformed position embedding are first converted to the same higher dimensional space using a feed-forward network and summed to obtain the input embedding.

The architecture involves a decoder and encoder as specified by [8]. Each Multi-head self-attention layer has 5 heads. The encoder contains 5 encoder blocks and the decoder consists of 3 decoder blocks. Each encoder-block and decoder-block are as specified in [8].

2.4. Temporal Convolution Network

Research on the Convolution Sequence-Sequence Model portrayed that convolution kernels can capture long-term dependencies and can model spatiotemporal dependencies without the need of several steps ($O(logn)$ steps) in the past to determine the future time steps to forecast weather data. Temporal convolutional networks (TCNs) use convolutions to model patterns over time. They are better than both LSTMs and RNNS at capturing long-range dependencies. The core concept of temporal Convolution Networks involves two main concepts known as causal convolution and receptive field. The causal convolution is when at the specific time step the output uses only the specific time step and the past time steps. The subsequent time steps will use the current input and the past time steps. The concept of the receptive field is to make the network use all the inputs prior to the current input so that it captures the maximum patterns. This is achieved by calculating the appropriate padding using the dilation and the kernel size. The TCN architecture mainly consists of several Temporal Blocks and each Temporal block has 2 convolution layers. The model

is trained on sequential data and that can be prepared in two ways for this TCN architecture. The two approaches used for the TCN include the n-m approach and the (n+k)-(k+m) approach where n is past time steps m is the future time steps and k is the overlap of the input and output time steps. Here n-m approach is when n is all past data and m is all future data and there is no overlap in the predictions of the time step. The (n+k)-(k+m) approach which generates sequences based on the overlap of predictions is in theory better as it uses more of the past data to make the predictions. Additionally, data can be highly parallelized similar to transformers, allowing faster training. This architecture is good for the case of forecasting but current TCN architecture may have tended to over-fit at a quite faster rate. This can be attributed to the edge effect as the beginning set of sequences and patterns are not captured. This configuration may not be as effective when the auto-regressive prediction is performed.

2.5. Temporal Convolution Network-Modified

The modification proposed to the existing TCN model involves two main components: Adding extra convolution bottleneck layers and a modified parameter loss function that will be discussed in the experimental section. The proposed modified architecture aims to achieve reduced computational complexity, mitigating overfitting as bottleneck acts as a form of regularisation and Reduced edge effect. This is a experimentation network and a potential suspects problem is that the network might overfit towards one feature and this is controlled by a new loss function. The modified TCN architecture is displayed in Figure 3.

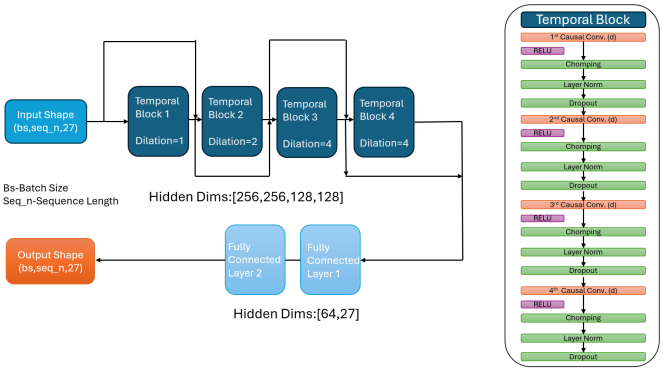

Figure.3. TCN Modified Architecture

3. Experiments and Results

3.1. Evaluation Policy

In terms of training, All architectures were run for 100 epochs, a batch size of 64, and training losses, validation losses, GPU memory usage, and number of parameters were computed for the current dataset. The best model is based on the least testing loss and is used to test the model's performance in an auto-regressive manner[9]. The definition of auto-regressive in this context is defined as using the predictions as the input to predict further into the future. This will enable the model to look further down the future. All architectures were trained on 4GB Nvidia RTX 4080 and A6000 GPUs using CUDA. The training data is split into training and validation sets, with 20% of the training data used for validation to monitor the model's performance and prevent over-fitting. The Adam optimizer is employed to minimize the mean squared error (MSE)/Mean Absolute Error (MAE) loss function. All the parameters were trainable and none of the preexisting weights were used for any model.

3.2. Performance Metrics and Optimizers

Mean Squared Error(MSE) in (1) and Mean Absolute Error(MAE) in (2) are the two loss functions that are used to train the networks/architecture separately. This is performed because MSE is particularly sensitive towards outliers and that may lead to a focus on overfitting that we want to mitigate. MSE also has larger gradients and that does not seem to be the issue with MAE. Potentially using MAE might also lead to underfitting and not correctly capturing the patterns. To test the best loss for the weather forecasting data both losses are utilised.Two different types of regularizing techniques are used to prevent overfitting - a dropout layer and l2 regularizer.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

We use the Adam optimizer to examine the effect of different learning rates (0.001 and 0.0001) on model training. The argument for a higher learning rate of 0.001 is that while it can speed up convergence in the early stages of training, it might also cause stability problems or overshoot minimums, particularly in complicated models. On the other hand, a slower but more stable convergence is anticipated with a lower learning rate of 0.0001, which may improve generalization on unobserved data and lessen the chance of overshooting. The experiments involved training and testing four different deep learning architectures: Recurrent Neural Networks (RNNs), Long-Short Term Memory networks (LSTMs), Transformers, and Temporal Convolutional Networks (TCNs), including a modified version of TCN. Each model was trained on the Jena climate dataset, which includes over a decade of climate measurements at 10-minute intervals.

The models were evaluated for their ability to predict six key weather variables: air pressure, temperature, relative humidity, air density, CO2 concentration, and precipitation. The performance of each model was tested at different forecast horizons: 3, 8, 16, and 24 hours into the future.

3.3. Modified Loss Function - MAE

The loss function plays a vital role in the network as it determines how well the network captures the distribution of the input features. The traditional approach towards training a network involves calculating MSE or MAE over all the features and doing back-propagation on that. The stated loss functions are what is used to train the network except for the modified TCN network. The modified TCN network uses a modification we proposed to the MAE loss function. This is achieved by adding weighted hyper-parameters[10] for all features with a specific distribution towards the 6 specified features and the rest of the features will have a fixed set of constant values. The hyper-parameters of the 6 features can be unevenly distributed based on the models' performance for each feature.

The Weighted Mean Absolute Error (MAE) is defined as:

$$\text{Weighted MAE} = \sum_{i=1}^{m} \frac{1}{n} \sum_{j=1}^{n} w_{Mi} |y_j - \hat{y}_j| \tag{3}$$

$$w_M = w_a + w_b \tag{4}$$

where w_M is the combined weight from two different sets of features, w_a and w_b, each contributing to the total weight applied to the error calculation. The weight w_a (5) is a list of weights corresponding to the total 6 features, denoted as:

$$w_a = [w_{a1}, w_{a2}, \ldots, w_{a6}] \tag{5}$$

The weight w_b (6) is a list of weights corresponding to the leftover features, which can vary in number, denoted as:

$$w_b = [w_{b1}, w_{b2}, \ldots, w_{bk}] \tag{6}$$

The modified TCN network utilizes this form of loss function to train and evaluate the network.

3.4. RNN and LSTM Analysis

As discussed earlier in the RNN section the RNN is a baseline model and we generally did not expect the model to perform at a high level compared to the SOTA architecture. As expected the RNN model performs the worst among all the models owing to the vanishing gradients and its loss of temporal memory over time. The RNN is trained on 64 units as an initial layer using the ReLU function. The overfitting can be avoided on an RNN to an extent by using linear layers and balancing its complexity.

Based on the findings, we could say that the LSTM model's better architectural features over RNNs are what allow it to function successfully over shorter forecast horizons. LSTMs are developed to solve the vanishing gradient issue, and they can retain knowledge for extended periods of time without seeing a discernible decline in accuracy.

However, when compared to TCN we can see that the LSTM performs worse, and that can be because LSTMs are designed to handle long-term dependencies better than RNNs, but they struggle with very long sequences due to the recurrent nature of their processing. TCNs, on the other hand, utilize dilated convolutions that allow them to have a much larger receptive field, which helps them look back at a broader range of input data in a single step.

Moreover, when it comes to minimizing significant errors over forecast horizons, LSTM models do better than Transformers overall in Mean Squared Error (MSE), but they fall short in Mean Absolute Error (MAE). This implies that, in contrast to Transformers, LSTMs perform less consistently across all predictions even while they are quite good at eliminating major errors. This discrepancy can result from LSTMs' sequential processing architecture, which limits its capacity to dynamically weigh the components of an input sequence. In contrast, Transformers' self-attention mechanism enables parallel processing and more effective handling of temporal dependencies.

In terms of the LSTM architecture, to optimize performance the model's parameters were tuned. 'Relu' activation function introduced non-linearity, which was crucial for learning complex patterns. Additionally, dropout layers with a rate of 0.2 were used to prevent overfitting. Lastly, the use of the Adam optimizer with a learning rate of 0.002 helps the model to adapt well across different data types. The model was trained for 100 epochs to strike a balance between achieving optimal performance and avoiding overfitting of the model.

In conclusion, the LSTM model demonstrates robust performance for short to medium-term weather forecasting, as evidenced by the low MSE and MAE values for forecasting these times. However, the increase in error metrics at the 24-hour forecast mark highlights the challenges LSTM faces in long-term weather prediction.

3.5. Transformer Analysis

Overall, transformers performed better than RNNs, but worse than LSTMs and TCN variants. This can observed through the MSE and MAE values shown in Table 2 and Table 3. As explained in the approach section, transformers are extremely data-hungry and take much longer to train. One of the primary reasons for this is that transformers have no inductive bias and make no assumptions about the inherent structure of the data. Therefore, they require significantly more training time and data. After much exper-

Model	MAE @ 3hr	MAE @ 8hrs	MAE @ 16hrs	MAE @ 24hrs
RNN	2.54E-01	4.6E-01	8.3E-01	9.8E-01
LSTM	7.89E-01	8.03E-01	8.29E-01	8.62E-01
Transformer	2.1E-01	2.61E-01	4.21E-01	4.89E-01
TCN	1.45E-01	2.83E-01	4.44E-01	5.68E-01
TCN-modified	**1.07E-01**	**2.08E-01**	**3.27E-01**	**3.84E-01**

Table 1. MAE comparison for different hours for various deep learning models for time-series forecasting.

Model	MSE @ 3hr	MSE @ 8hrs	MSE @ 16hrs	MSE @ 24hrs
RNN	1.39E+00	1.756E+00	1.8E+00	8.04E+00
LSTM	1.17E+00	1.28E+01	1.32E+00	2.63E+00
Transformer	3.12E+00	4.42E+00	11.34E+00	15.23E+00
TCN	3.8E-01	6.99E-01	1.211E+00	1.907E+00
TCN-modified	**3.22E-01**	**4.91E-01**	**6.71E-01**	**8.41E-01**

Table 2. MSE comparison for different hours for various deep learning models for time-series forecasting.

imentation, it was observed that hyperparameters of 5 encoders, and 3 decoders worked best. Several dropout values were used of which 0.1 worked best. No data augmentation was performed due to the large size of the dataset. The model was trained to about 60 epochs and further training combined with data augmentation would have resulted in better results. It is possible that transformers could outperform LSTMs, however, the LSTMs assume sequential data, thus making them faster to train even without parallelization. It is unclear without further computational resources and training data if the current transformer architecture has the potential to outperform the TCN variants discussed below. However, there is much scope with this mechanism and perhaps further architectural changes could allow this to produce the best results.

3.6. TCN and TCN-Modified Analysis

The TCN architecture and The TCN modified architecture use the hyperparameters mentioned in Table 1 and this was achieved by comparing the loss vs epoch curves for all such scenarios. Before comparing the results across all other models it is important to understand that as the forecast horizon increases errors tend to accumulate. The TCNs and the modified TCN with the novel loss function outperformed the other models across the MAE metric for all forecast horizons. As explained in the prior sections the only 2 key changes between the TCN and modified TCN is the change in convolution layers and the loss function. The first trend to notice from Table 3 that compares evaluation MAE values is that for the modified TCN the accumulation of errors is minimal compared to the error accumulation of the classic TCN model. This is attributed to the weighted loss function that trained the model and captured the important features that we wanted to predict in an autoregressive manner. The second important trend is that the bottleneck layer implemented in the Temporal Block significantly reduced the number of parameters. The third trend is that the classic TCN overfits the existing data faster than the TCN-modified model. The above analysis concludes the most suitable model for our case of weather forecasting is the TCN-modified network that fits well with the data and understands the forecasting patterns for the specific features we wanted to predict. In Figure 4 we have forecasted weather prediction data for the 6 features.

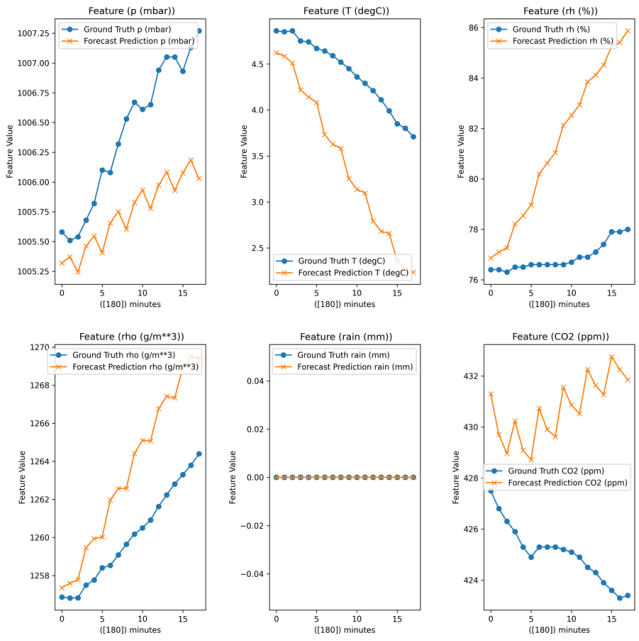

Figure 4. Forecast (3 hours) - 6 Feature Distribution

References

[1] M. G. Schultz, C. Betancourt, B. Gong, F. Kleinert, M.Langguth, L. H. Leufen, A. Mozaffari, and S. Stadtler.

Can deep learning beat numerical weather prediction? Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 379, 2011.

[2] Yuejian Zhu. Ensemble forecast: *A new approach to uncertainty and predictability*. Advances in Atmospheric Sciences, Physical, and Engineering Sciences, 781–788, 2005.

[3] Degbelo, A., Kuhn, W. *Spatial and temporal resolution of geographic information: an observation-based theory*. Open Geospatial Data, Software and Standards, 3, 12 (2018). https://doi.org/10.1186/s40965-018-0053-8.

[4] Gong, B., Langguth, M., Ji, Y., Mozaffari, A., Stadtler, S., Mache, K., & Schultz, M. G. *Temperature forecasting by deep learning methods*. Geoscientific Model Development, 15(23), 8931–8956 (2022).

[5] Othman, W.; Hamoud, B.; Kashevnik, A.; Shilov, N.; Ali, A. *A Machine Learning-Based Correlation Analysis between Driver Behaviour and Vital Signs: Approach and Case Study*. Sensors 2023, 23, 7387. https://doi.org/10.3390/s23177387.

[6] M. Schuster and K. K. Paliwal, *"Bidirectional recurrent neural networks,"* in IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, Nov. 1997, doi: 10.1109/78.650093.

[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 15, 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł.

[9] Kaiser, and I. Polosukhin. *Advances in Neural Information Processing Systems 30, Curran Associates, Inc., (2017)* Burant CJ. A Methodological Note: An Introduction to Autoregressive Models. Int J Aging Hum Dev. 2022 Dec;95(4):516-522. doi: 10.1177/00914150211066554. Epub 2021 Dec 6. PMID: 34866432.

[10] Rengasamy D, Jafari M, Rothwell B, Chen X, Figueredo GP. Deep Learning with Dynamically Weighted Loss Function for Sensor-Based Prognostics and Health Management. Sensors (Basel). 2020 Jan 28;20(3):723. doi: 10.3390/s20030723. PMID: 32012944; PMCID: PMC7038523.

4. Work Division and Appendix

Student Name	Contributed Aspects	Details
Vaibhav Malhotra	RNN implementation and Report	Scraped the dataset for this project and trained the RNN on the data. Implemented RNN hyperparameter tuning to improve the model.
Shashank Garikipati	Transformer Implementation and Report	Modelled transformer from scratch and trained encoder and decoder. Hyperparameter tuning with a number of encoders/decoders. compared effects of position embedding. Analyzed dataset for feature analysis for optimal model training
Koushik Karan Geetha Nagaraj	TCN implementation and Report	Scraped the dataset for this project and trained the TCN on the dataset. Implemented the modified TCN network with the loss and documented the results of the forecast.
Pranav Sharma	LSTM implementation and Report	Trained the LSTM model and analyzed the results. Analyzed the effect of a number of nodes in a hidden state. Implemented Convolutional LSTM. Analyzed dataset for feature analysis for optimal model training.

Table 3. Contribution Table

Model	**Num. Params**	**GPU Memory**	**Hidden Dim.**	**Train Loss MAE**	**Val Loss MAE**	**Train Loss MSE**	**Val Loss MSE**
RNN	212,757	1.03 GB	[512, 256, 128, 64, 32]	2.151E-01	2.775E-01	1.394E-01	2.193E-01
LSTM	520,021	2.12 GB	[128, 256, 128, 64, 64]	2.328E-01	2.479E-01	1.276E-01	1.824E-01
Transformer	524,521	4.32 GB	[500 for each encoder/decoder]	2.11E-01	2.66E-01	1.32E-02	1.62E-01
TCN	17,61,563	2.15 GB	[512,512,256,256]	8.823E-02	9.456E-02	8.741E-02	10.321E-02
TCN-modified	5,39,192	2.15 GB	[512,512,256,256]	3.686E-02	4.273E-02	7.98E-02	8.16E-02

Table 4. Comparative performance metrics of various deep learning models for time-series forecasting.

Symbol	Unit	Variable
Date Time	DD.MM.YYYY HH:MM (MEZ)	Date and time of the data record (**end**)
p	mbar	air pressure
T	°C	air temperature
Tpot	K	potential temperature
Tdew	°C	dew point temperature
rh	%	relative humidity
VPmax	mbar	saturation water vapor pressure
VPact	mbar	actual water vapor pressure
VPdef	mbar	water vapor pressure deficit
sh	$g\,kg^{-1}$	specific humidity
H2OC	$mmol\,mol^{-1}$	water vapor concentration
rho	$g\,m^{-3}$	air density
wv	$m\,s^{-1}$	wind velocity
max. wv	$m\,s^{-1}$	maximum wind velocity
wd	°	wind direction
rain	mm	precipitation
raining	s	duration of precipitation
SWDR	$W\,m^{-2}$	short wave downward radiation
PAR	$\mu mol\,m^{-2}\,s^{-1}$	photosynthetically active radiation
max. PAR	$\mu mol\,m^{-2}\,s^{-1}$	maximum photosynthetically active radiation
Tlog	°C	internal logger temperature
CO2	ppm	CO_2-concentration of ambient air

Figure.5. Data Features

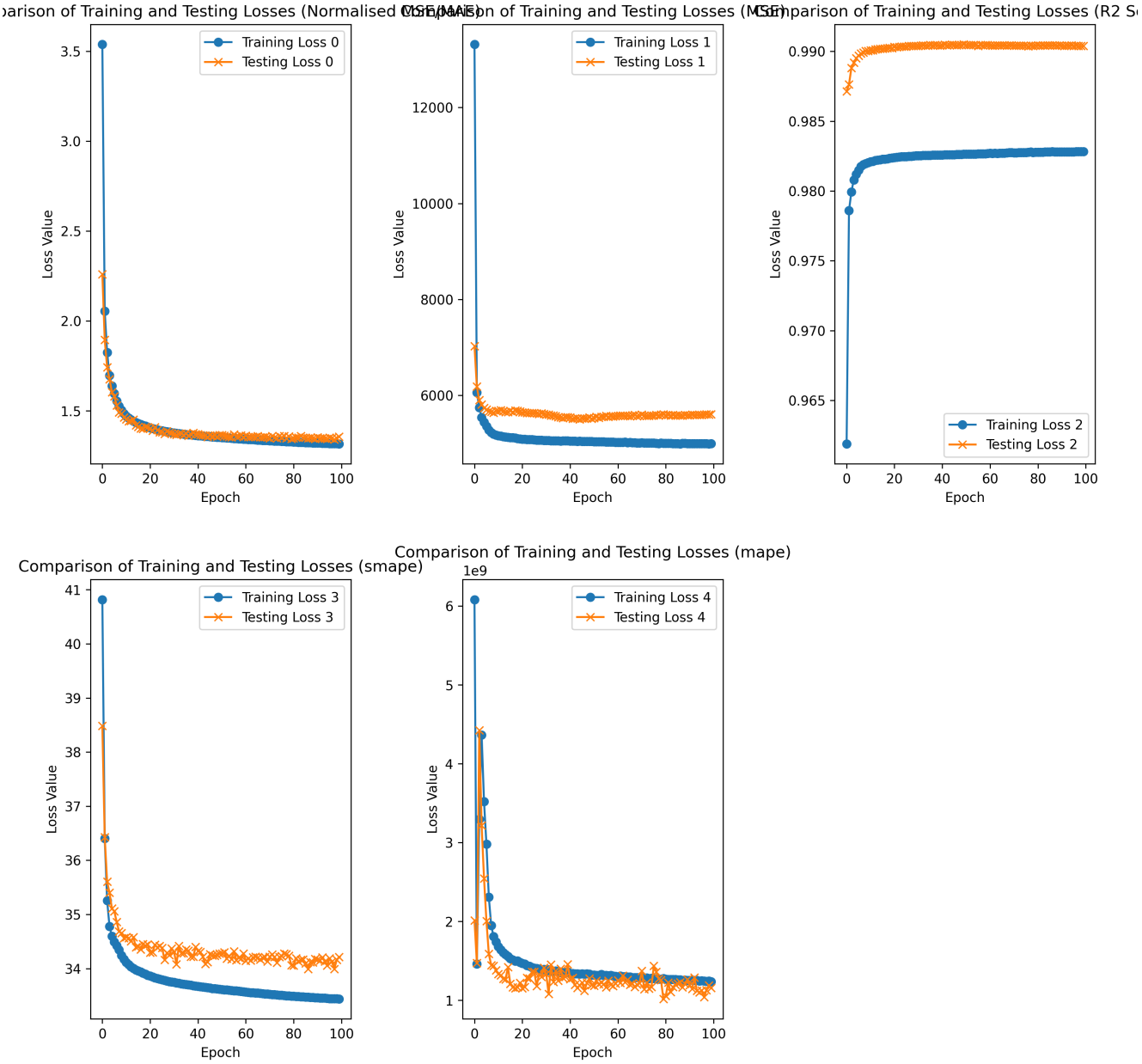

Figure.6. TCN Modified Architecture Losses Plot

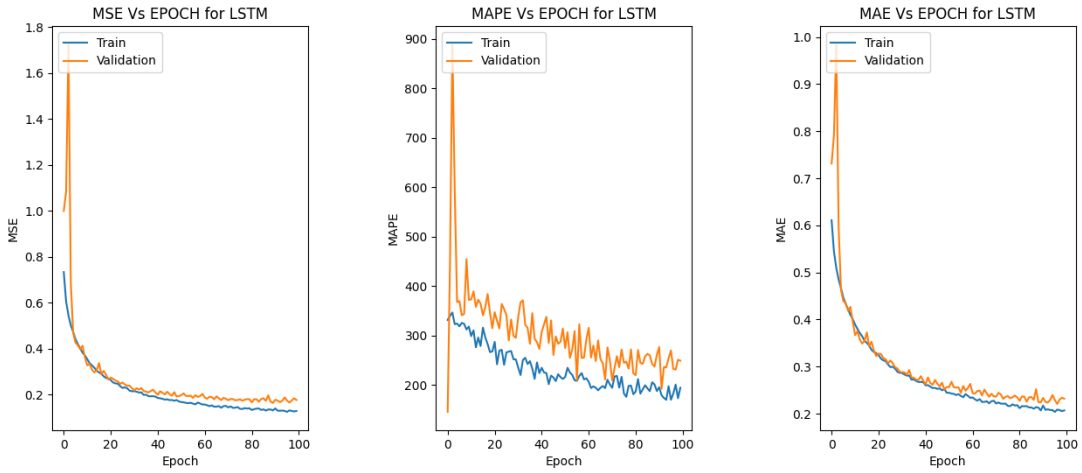

Figure.7. LSTM Architecture Losses Plot

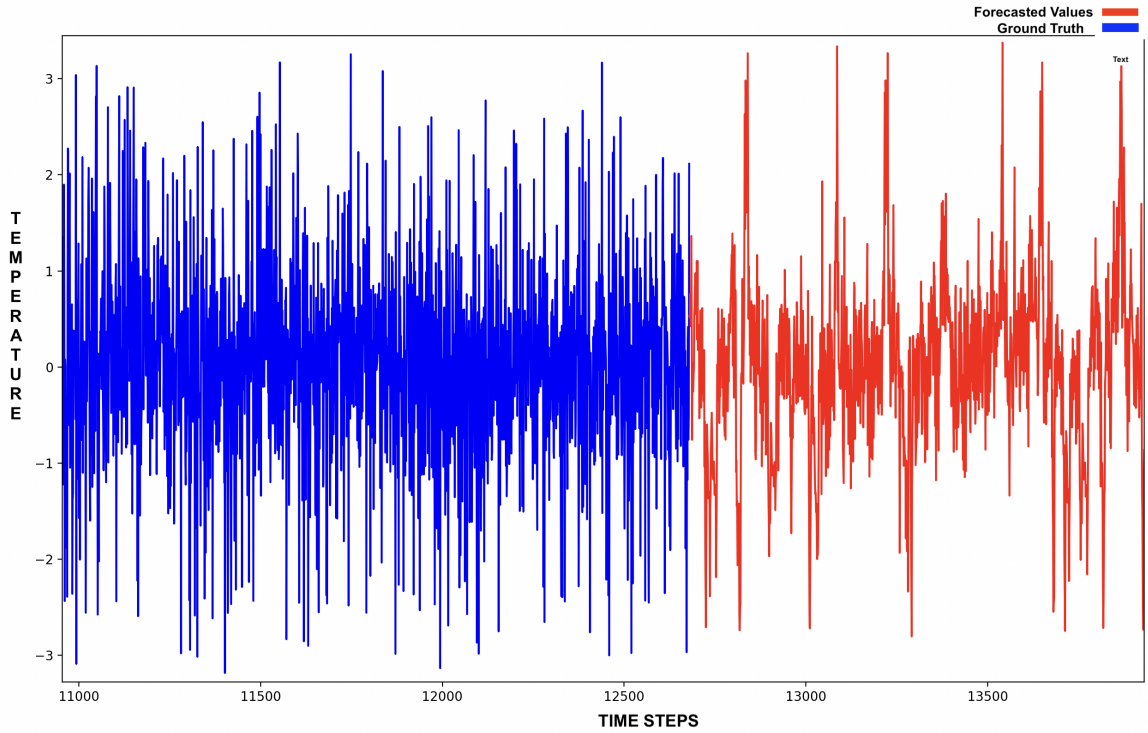

Figure.8. Plot of Ground Truth vs Forecasted Temperature for the LSTM model