Course Project
CSCI.635.01-02 - Intro to Machine Learning

Final Project Report

Instructor: Prof. Eduardo Lima

Group Members: Sterling Alexander , Manvith Dayam , Pankhuri Shukla , Divyansh Vaidya

Dept. of Computer Science

Rochester Institute of Technology

1. Task Definition, Evaluation Protocol, and Dataset

The task is to use deep learning models to perform weather forecasting over India. This is achieved by predicting meteorological variables, such as temperature and precipitation, based on past data. The task is structured as a regression problem where the model attempts to predict continuous values (e.g., temperature, precipitation amounts)

Dataset: IMDAA Regional Reanalysis Dataset, Available from the IMDAA Reanalysis Portal. Made by India Meteorological Department (IMD), National Centre for Medium Range Weather Forecasting (NCMRWF), Indian Institute of Tropical Meteorology (IITM)

Data Variables: The key variables used for prediction include:

a. Geopotential height at 500 hPa (H500),
b. Temperature at 850 hPa (T850),
c. Temperature at 2m (T2m),
d. Six hourly accumulated precipitation (TP6h).

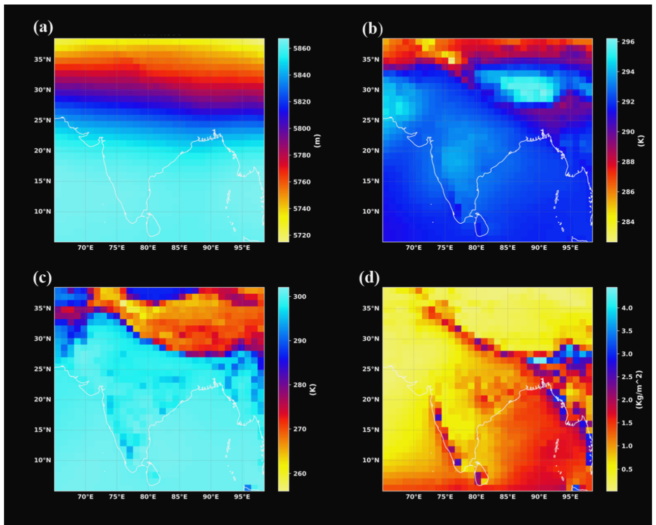


Figure 1: Spatial distribution of average (a) H500, (b) T850, (c) T2m, and (d) TP6h.

The dataset is divided into three distinct periods for the purpose of training, validation, and testing the model. The training data spans from the year 1990 to 2017, providing a comprehensive dataset for the model to learn from. The year 2018 is designated as the validation period, allowing for the fine-tuning of the model's parameters and helping to prevent overfitting. Finally, the test data covers the years 2019 to 2020, enabling the assessment of the model's performance on unseen data.

Metrics for evaluating the models:

1. RMSE (Root Mean Square Error): Measures the standard deviation of the prediction errors.
2. MAE (Mean Absolute Error): Measures the average magnitude of errors in a set of predictions.
3. ACC (Anomaly Correlation Coefficient): Measures the correlation between predicted and actual anomalies, indicating how well the model captures the anomaly pattern.

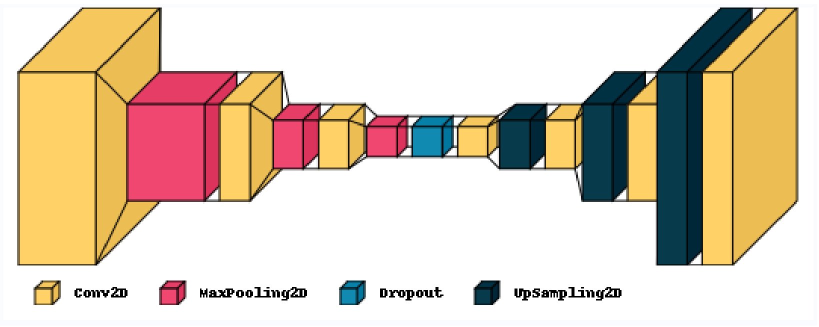
2. Neural Network / Machine Learning Model

Standard CNN Model

A convolutional neural network (CNN) was initially implemented to handle the weather data prediction task. The CNN architecture was as follows:

- Convolutional Layers: The network began with several convolutional layers to extract features from the input data. Each convolutional layer was followed by batch normalization and a swish activation function to introduce non-linearity and stabilize training.
- The swish activation function combines the input x with its sigmoid function in the following equation: $Swish(x) = x \cdot \sigma(x)\ where\ \sigma(x) = \frac{1}{1+e^{-x}}$. This was initially preferred due to better gradient flow and less vanishing gradient problems compared to ReLU; according to Kılıçarslan et al (2021), the algorithm is best suited to image processing but regularly achieves superior results over ReLU regardless.
- Pooling Layers: Max pooling layers were employed to reduce the spatial dimensions of the data and capture important features.
- Dense Layers: After flattening the output of the convolutional layers, several dense layers were used to predict the final output.
- Batch Normalization: Batch normalization (BatchNorm) is a technique used to improve the training of deep neural networks by normalizing the inputs to each layer.

The original CNN model achieved reasonable performance on the evaluation metrics, indicating its effectiveness in predicting the weather data.



Modifications to the CNN Model

Activation Function: The activation function was changed from the Swish function to the Scaled Exponential Linear Unit (SELU) function, defined as follows:

- $selu(x) = \lambda x\ if\ x > 0$
- $selu(x) = \alpha e^{x} - \alpha\ if\ x \leq 0$
- λ and α are predefined constants; this requires specific weight initialization but is self-normalizing: because it helps to keep the mean and variance of the inputs to each

layer normalized throughout the network, it simultaneously helps to stabilize the training process and ensures faster convergence. Activation functions introduce non-linearity into the model where a neural network would otherwise be equivalent to a linear regression model, regardless of its depth, and would not be able to handle complex data patterns.

Removal of Batch Normalization: Adding BatchNormalization can help in several ways:

- It can speed up training by reducing internal covariate shift.
- It can act as a regularizer, potentially reducing the need for Dropout in some cases.
- It can make the model less sensitive to weight initialization.
- It can often use higher learning rates without risking the training becoming unstable. This can speed up the training process significantly.

However, the switch to the SELU activation function, providing self-normalization, reduces the need for batch normalization. In practice, the removal of batch normalization was found to yield better results, rendering its removal as an improvement over the original.

ConvLSTM Model

A ConvLSTM (Convolutional Long Short-Term Memory) model was then developed to capture spatiotemporal patterns in the weather data. The ConvLSTM model integrates convolutional layers with LSTM layers to handle sequences of spatial data. Similar approaches have achieved effective results, such as a study by Hou et al (2022) demonstrating the use of a CNN-LSTM in forecasting. The CNN-LSTM, separating the spatial and temporal components of the ConvLSTM model into two stages (CNN Layers and LSTM Layers), was an effective predictor of hourly air temperature in China given data from 2000-2020.

The architecture of the ConvLSTM model included:

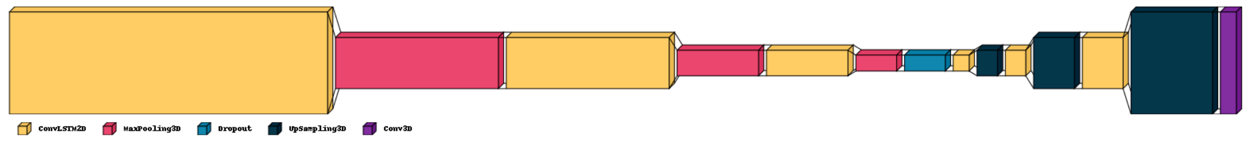
- ConvLSTM2D Layers: These layers combined convolutional operations with LSTM units to learn spatiotemporal features from the input data.
- Pooling and UpSampling Layers: Max pooling and upsampling layers were used to manage the spatial dimensions of the data and recover the original input size.
- Dropout: Applied to reduce overfitting by randomly setting a fraction of input units to zero during training.
- Batch Normalization: as for the CNN model, batch normalization was implemented, normalizing the activations of each layer to have zero mean and unit variance.

The performance of the original ConvLSTM model was not satisfactory, showing higher errors compared to the CNN model.

Modified ConvLSTM Model

To improve the performance, the ConvLSTM model was modified with different layers and architecture:

- Activation Function: Switched to the SELU (Scaled Exponential Linear Unit) activation function to improve training stability.
- Removal of batch normalization: Similarly to the CNN, interaction with activation functions as well as empirical performance observations warranted removal of batch normalization from the model.



Loss Metrics

The CNN and LSTM models were evaluated using the following metrics:

- RMSE (Root Mean Squared Error): Measures the standard deviation of the residuals, indicating how much the predictions deviate from the actual values.
- MAE (Mean Absolute Error): Represents the average magnitude of the errors in the predictions.
- ACC (Accuracy): Represents the proportion of correctly predicted values out of the total predictions.

3. Experiment

Research Question: Our purpose from this experiment is to learn more about the behavior of the original CNN and convLSTM models mentioned in the paper, and with different types of architecture and hyperparameters, to try to improve the loss function and evaluation metrics obtained in the original research to see if a better tuned convLSTM can perform equal to or better than CNN due to its ability of being a spatio temporal model.

Design:
Hypothesis: Changes made to hyperparameters and the overall model structure will improve their performance on the given dataset, reducing the error obtained and increasing anomaly correction, and the performance of convLSTM will be equal to or better than CNN at weather prediction.

Independent variables (Experimental Settings):

1. Activation function
2. Number of layers
3. Number of filters
4. Filter size

Control Variables (Biases and Modeling Assumptions):

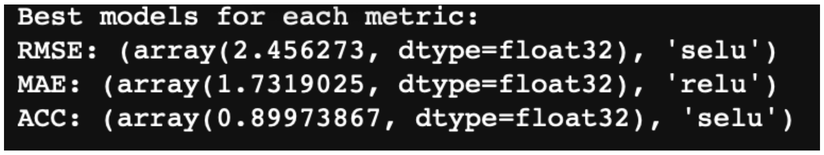
1. Number of epochs will remain constant (10 for CNN, 5 for convLSTM)
2. Dataset will stay the same
3. Learning rate, optimizer (Adam), number of batches will remain the same
4. Type of pooling will remain MaxPooling

Dependent variables (Result Analysis):

1. Training loss metrics curve over number of epochs
2. Evaluation metrics mentioned above (RMSE, MAE, ACC)

Methodology:
Our baseline for this experiment is the use of the original BharatBench dataset for weather prediction and the existing implementation on CNN and convLSTM provided by the original authors for comparison. In order to improve the accuracy of the model, initial focus remained on evaluation metrics of RMSE, MAE and ACC, as used by the authors. We performed the following changes in order to reach an optimal solution upon continuous adjustments:

<u>*Modified CNN:*</u>
Starting with an analysis of which would be the most optimal activation function for our model (originally 'swish'), we ran 10 epochs for each activation type on the original CNN model and measured the evaluation metrics. Activation 'selu' was the best out of all, displayed by an overall metric evaluation.



```
Best models for each metric:
RMSE: (array(2.456273, dtype=float32), 'selu')
MAE: (array(1.7319025, dtype=float32), 'relu')
ACC: (array(0.89973867, dtype=float32), 'selu')
```

The original paper mentions batch normalization as a way to improve training of deep neural networks by normalizing input to each layer, which was evaluated in a separate file but not included in the layer architecture due to bad performance metrics. An attempt was also made to change the number of batches to 128 and 64 from 32, which could potentially increase the stability of the optimization, but it also worsened the errors slightly when comparing activations again.

Normally such problems do not need padding changes, so our main focus was on the filters and their dimensions. Increasing the number of filters means we have more parameters to train on, but risks overfitting the data. Layer manipulation was done by changing the number of layers while tuning the filter size, default stride, as follows:

1. Decreased 1 inner layer: *Slight Improvement*
2. Decreased 1 more inner layer and increased filters in layer 2 to 64: Time taken increased to 12 ms/step, *Slight Improvement*
3. Increased number of filters in layer 1 to 512 to recognize more complex patterns: *Slight Improvement*
4. Increased 1 inner layer *[Best Metrics]*
5. Increased learning rate to 1e-04: Time taken was approximately 17 ms/step *[Best Graph] [Chosen Model]*
6. Same layers, increased learning rate to 1e-03, *Big Downgrade*
7. Same layers, decreased learning rate back to 1e-04 and decreased drop rate from 0.3 to 0.2, *Slight Downgrade*

No. 4 had the best evaluation metrics results, but upon analysis of their graphs, we noticed the graph for No. 5 for loss during training was significantly better, so that was chosen as our optimal model. Modified CNN is also trained for 10 epochs, to compare to original.

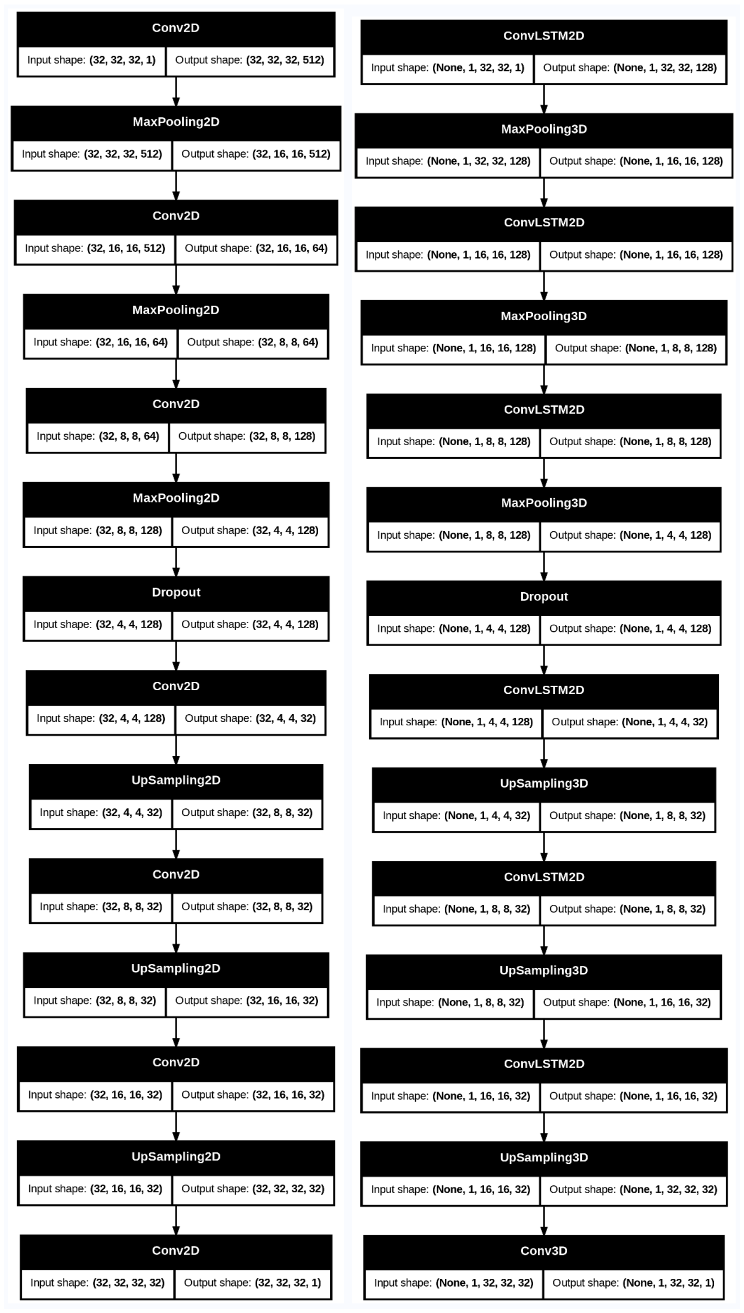
Modified convLSTM:

The same activation function was determined to be the best for convLSTM ('selu'). The model was trained for 1 epoch due its complexity and time elapsed per epoch step being comparatively higher than CNN. Changes to the model architecture and hyperparameters were made as follows:

1. Layers with 512,64,64,32 number of filters: *Increased time significantly*
2. Layers with 128,64,64,32 number of filters: *Reduced time, reduced accuracy*
3. Same layers, learning rate increase to 1e-04: *Reduced time, increased accuracy*
4. 1 less layer: *Reduced time, increased accuracy*
5. Layers with 128,128,128 number of filters, drop rate increased to 0.3: 58 ms/step time elapsed, *Increased time, Increased accuracy [Chosen Model]*

Trying to increase the batches here worsened error and increased the time taken significantly, so it remained 32.

Attempts to try a new data variable from the dataset:
Original paper noted variable T850 as the best, named TMP_prl. Variables of APCP_sfc and TMP_2m were also tested against both new models. Comparing the models again for the best activation for these new variables, the answer remained ‘selu’.


Conv2D
Input shape: (32, 32, 32, 1) | Output shape: (32, 32, 32, 512)
MaxPooling2D
Input shape: (32, 32, 32, 512) | Output shape: (32, 16, 16, 512)
Conv2D
Input shape: (32, 16, 16, 512) | Output shape: (32, 16, 16, 64)
MaxPooling2D
Input shape: (32, 16, 16, 64) | Output shape: (32, 8, 8, 64)
Conv2D
Input shape: (32, 8, 8, 64) | Output shape: (32, 8, 8, 128)
MaxPooling2D
Input shape: (32, 8, 8, 128) | Output shape: (32, 4, 4, 128)
Dropout
Input shape: (32, 4, 4, 128) | Output shape: (32, 4, 4, 128)
Conv2D
Input shape: (32, 4, 4, 128) | Output shape: (32, 4, 4, 32)
UpSampling2D
Input shape: (32, 4, 4, 32) | Output shape: (32, 8, 8, 32)
Conv2D
Input shape: (32, 8, 8, 32) | Output shape: (32, 8, 8, 32)
UpSampling2D
Input shape: (32, 8, 8, 32) | Output shape: (32, 16, 16, 32)
Conv2D
Input shape: (32, 16, 16, 32) | Output shape: (32, 16, 16, 32)
UpSampling2D
Input shape: (32, 16, 16, 32) | Output shape: (32, 32, 32, 32)
Conv2D
Input shape: (32, 32, 32, 32) | Output shape: (32, 32, 32, 1)
ConvLSTM2D
Input shape: (None, 1, 32, 32, 1) | Output shape: (None, 1, 32, 32, 128)
MaxPooling3D
Input shape: (None, 1, 32, 32, 128) | Output shape: (None, 1, 16, 16, 128)
ConvLSTM2D
Input shape: (None, 1, 16, 16, 128) | Output shape: (None, 1, 16, 16, 128)
MaxPooling3D
Input shape: (None, 1, 16, 16, 128) | Output shape: (None, 1, 8, 8, 128)
ConvLSTM2D
Input shape: (None, 1, 8, 8, 128) | Output shape: (None, 1, 8, 8, 128)
MaxPooling3D
Input shape: (None, 1, 8, 8, 128) | Output shape: (None, 1, 4, 4, 128)
Dropout
Input shape: (None, 1, 4, 4, 128) | Output shape: (None, 1, 4, 4, 128)
ConvLSTM2D
Input shape: (None, 1, 4, 4, 128) | Output shape: (None, 1, 4, 4, 32)
UpSampling3D
Input shape: (None, 1, 4, 4, 32) | Output shape: (None, 1, 8, 8, 32)
ConvLSTM2D
Input shape: (None, 1, 8, 8, 32) | Output shape: (None, 1, 8, 8, 32)
UpSampling3D
Input shape: (None, 1, 8, 8, 32) | Output shape: (None, 1, 16, 16, 32)
ConvLSTM2D
Input shape: (None, 1, 16, 16, 32) | Output shape: (None, 1, 16, 16, 32)
UpSampling3D
Input shape: (None, 1, 16, 16, 32) | Output shape: (None, 1, 32, 32, 32)
Conv3D
Input shape: (None, 1, 32, 32, 32) | Output shape: (None, 1, 32, 32, 1)

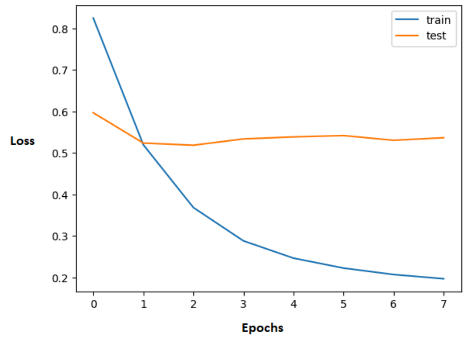
Modified CNN and Modified convLSTM

4. Experimental Results and Discussion

The results we obtained after fine tuning both the models were as follows:

Experiments for the CNN model:

Adding Batch Normalization:
Adding Batch Normalization after the Convolution layers resulted in decreased performance of the mode (increased Errors and decreased ACC). Also the loss is very high and increases with increase in epochs



Training loss curve with large error
(RMSE: 4.575, MAE: 3.166, ACC: 0.679)

Possible reasons why Batch Normalization might not have worked:

- Adding batch normalization increases the number of parameters and complexity of the model. For our specific dataset and task, this added complexity might have been unnecessary or even detrimental.
- The nature of weather data, which often involves time series and spatial correlations, might not benefit from the normalization that BatchNorm provides.
- Batch normalization can sometimes lead to overfitting, especially on smaller datasets.
- Batch normalization performance can be sensitive to batch size. If the batch size is too small, the estimates of mean and variance might not be accurate, which can degrade performance.

Batch normalization can allow for higher learning rates. After adding batch normalization, it is suggested to adjust the learning rates which might lead towards optimal performance. This venue can be considered for further experiments to improve the performance of the model.

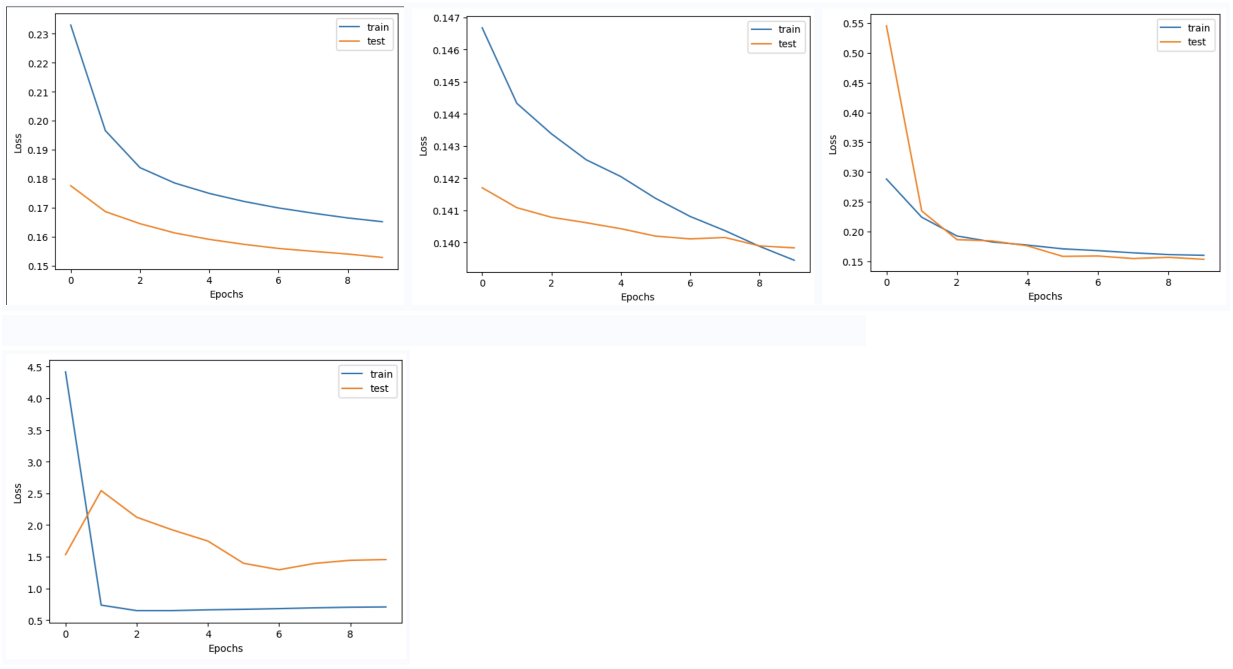
Changing Activation Functions:
We tested the model with various activation functions and found that the performance with ‘selu’ *(Scaled Exponential Linear Unit)* was the best. However, the **exponential** activation function faced gradient explosion. Increasing the batch size to 64 or 128 in an attempt to increase stability did not improve the gradient explosion issue.

Comparing activations for original CNN model with batch size 32

Models | Metrics	RMSE	MAE	ACC
relu	2.466	1.735	0.898
sigmoid	5.741	3.949	0.009
softmax	5.878	4.006	-1.396 e-06
softplus	2.596	1.834	0.886
softsign	2.631	1.847	0.884
tanh	2.518	1.774	0.894
selu	**2.365**	**1.671**	**0.906**
elu	2.502	1.762	0.895
exponential	nan	nan	nan
leaky_relu	2.564	1.799	0.890
relu6	2.538	1.786	0.891
silu	2.683	1.887	0.878
hard_silu	2.750	1.933	0.871
gelu	2.656	1.872	0.880
hardh_sigmoid	5.758	3.957	1.745 e-05
linear	2.484	1.753	0.896
mish	2.631	1.848	0.883
log_softmax	9.898	7.458	0.016
swish	2.694	1.895	0.877
hard_swish	2.691	1.897	0.877

Modifying Layers of the Model:
Changing the number of layers by either decreasing or increasing, we noticed CNN performs better with 2 layers before and after dropout (not counting pooling), with the best metrics, low errors and high ACC. This suggests that a simpler model architecture with fewer layers is more effective for our specific problem, possibly due to reduced overfitting and more efficient learning

of relevant features. However, the learning loss curves obtained by a 3 layer architecture after fine tuning seemed to fit better and converge. These readings were highly sensitive to learning rate changes, so it was challenging to not overfit to the data. Our findings indicate that a moderately deeper architecture with an appropriately high learning rate can enhance the model's ability to learn complex patterns efficiently, striking a balance between underfitting and overfitting. Increasing the number of filters improved the performance of the model slightly. Here are some graphs after the various changes mentioned:



Training loss curves for variations in layers: 2 layers, 3 layers (best metrics graph), 3 layers (most fitting graph, chosen model), 1e-03 learning rate curve (noisy)

<u>Adjusting number of batches:</u>

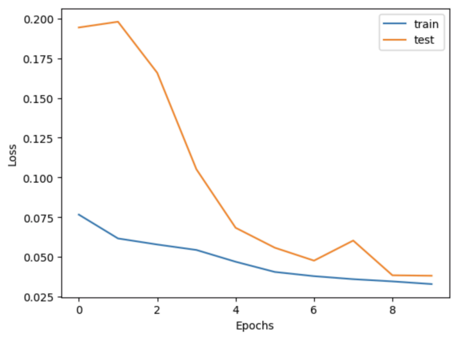
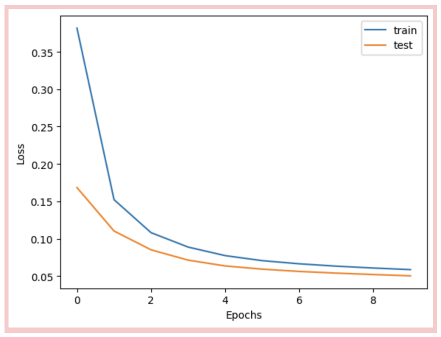
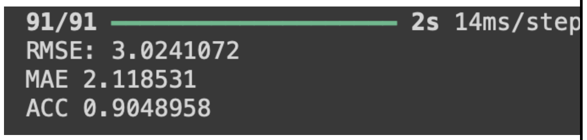
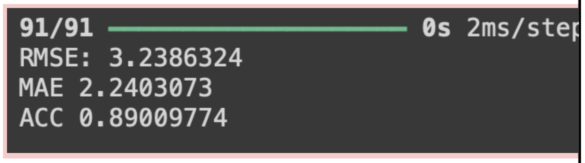
The batch size was increased from 32 to 128 to analyze its effect on the error metrics. Surprisingly, larger batch sizes led to a slight worsening of the error metrics. This could be due to less frequent updates to the model weights, resulting in slower convergence. Smaller batch sizes often provide a regularization effect and more noisy gradient updates, which can help escape local minima and improve generalization.

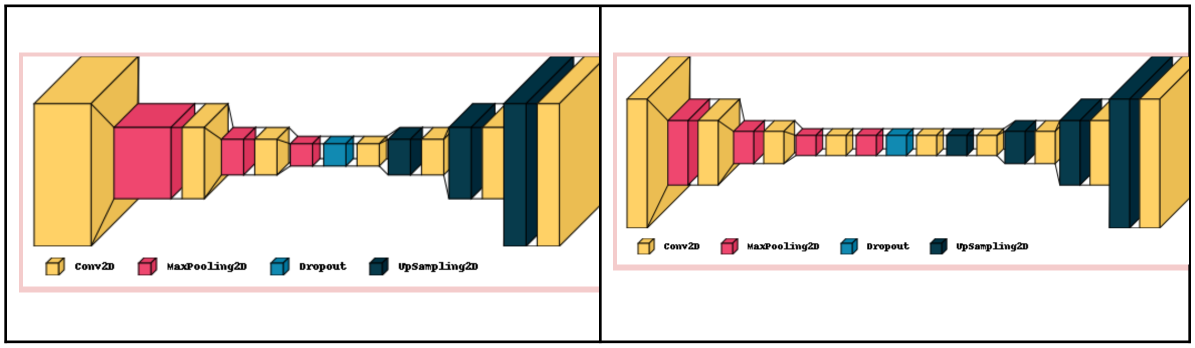
Table showcasing the recorded results after layer manipulation and hyperparameter tuning:

	RMSE	MAE	ACC
Original CNN	2.584	1.821	0.887

Adding Batch Normalization	4.575	3.166	0.679
New CNN 4 layers	2.318	1.627	0.910
New CNN 3 layers	2.289	1.607	0.912
New CNN 2 layers	2.269	1.588	0.914
New CNN 2 layers, more filters	2.248	1.576	0.915
New CNN 3 layers same lr	2.220	1.553	0.918
New CNN 3 layers with more filters in inner layer, higher lr	2.337	1.646	0.910
New CNN 3 layers with higher lr	2.334	1.685	0.910
New CNN 3 layer, higher lr, drop rate 0.2 instead of 0.3	2.338	1.649	0.910

Further, we tested the new model for TMP_2m variable from the dataset, and the below table showcases the results:

New CNN on TMP_2m	Old CNN on TMP_2m
	
 91/91 2s 14ms/step RMSE: 3.0241072 MAE 2.118531 ACC 0.9048958	 91/91 0s 2ms/step RMSE: 3.2386324 MAE 2.2403073 ACC 0.89009774



Experiments for the convLSTM model :

Adjusting Batch Size:

In our experimentation, increasing the batch size from 32 to 128 resulted in worsened error metrics and significantly increased the epoch time. Larger batch sizes lead to less frequent updates to the model weights, which can slow convergence and reduce the ability to escape local minima. Smaller batch sizes can add more noise in gradient updates, helping improve generalization and regularization.

Optimizing Learning Rate:

Original learning rate was very low at 1e-06, which hindered the training process. Increasing the learning rate to 1e-04 led to substantial performance improvements. A higher learning rate helps the model converge faster by making more significant updates to the weights during training, provided it is not too high to destabilize the training.

Experimenting with number of filters:

Initially, we configured the model with (512, 64, 64, 32) filters respectively across the four layers in both the encoder and decoder sections. This setup resulted in high runtime due to the large number of parameters. To reduce runtime, we adjusted the filters to (128, 64, 64, 32) respectively. Although this reduced the runtime, it also worsened the model's performance, likely due to insufficient capacity to capture the complexity of the data.

Reducing model complexity:

Removing one layer and using (128, 64, 64) filters in the remaining layers improved performance further. This suggests that the original four-layer configuration might have been overly complex, leading to potential overfitting or inefficient learning

Fine-tuning Filter Configuration and Dropout Rate:

We then modified the filters to (128, 128, 128) and increased the dropout rate from 0.2 to 0.3. This configuration achieved the best performance with an RMSE: 2.784, MAE: 1.958, ACC: 0.876. The increased dropout rate helped regularize the model, preventing overfitting and improving generalization. Despite the higher training time compared to the original setup, the performance gains justified the trade-off.

Further Increasing Filters:

Although increasing the number of filters beyond (128, 128, 128) led to improved performance, it came at the cost of significantly higher training time. This trade-off between performance and training time needs careful consideration depending on the application's requirements.

Findings:

Our experiments with the ConvLSTM model revealed that balancing the number of filters, batch size, and learning rate is crucial for optimizing performance. While higher filter counts and learning rates can improve accuracy and reduce errors, they also increase training time. Similarly, smaller batch sizes can help the model generalize better but may slow down training.

The best-performing model configuration was achieved with (128, 128, 128) filters, a dropout rate of 0.3, and a learning rate of 1e-04, providing a good balance between performance and training efficiency.

We then applied this optimal configuration to two other variables in the dataset. However, the model performed poorly on these variables, indicating that the configuration tuned for `tmp_prl` did not generalize well to other variables (`tmp_2m and aspc`). This emphasizes the need for individualized tuning to address the distinct characteristics of each variable. Each variable in the dataset may have unique characteristics and patterns that require tailored model configurations. Factors such as data distribution, underlying trends, and the complexity of temporal patterns can vary significantly between variables, impacting the model's ability to perform well in a general manner.

Possible Conclusion:

The evaluation metrics are not the sole indicator of which algorithm is better. Even though the training loss curve and the error metrics were better for our new model on dataset variable TMP_prl (the one finally used in the paper), evaluation on another metric TMP_2m gave a drastically different looking loss curve even though the error is still less in comparison. Moreover, APCP_sfc was giving very bad results and is excluded from this comparison. Although we were finally able to improve the overall performance of the model, the model needs to be tuned independently for all the variables of the dataset in order to perform better as they all might have different characteristics.

ConvLSTM’s performance did not change much for this new dataset variable, so using the original variable as intended here as well. Hence, the hypothesis that ConvLSTM would show comparable performance to CNN has failed.

Some possible next steps could be exploring Batch Normalization with higher learning rates and finetuning more parameters if the results seem to improve, trying different kinds of layers in the model, experimenting with different combinations of activation functions.

References

Choudhury, A., Panda, J., & Mukherjee, A. (2024). BharatBench: Dataset for data-driven weather forecasting over India. arXiv [Physics.Ao-Ph]. (http://arxiv.org/abs/2405.07534)

Michel Jose Anzanello, Flavio Sanson Fogliatto,Learning curve models and applications: International Journal of Industrial Ergonomics, Volume 41, Issue 5, 2011, Pages 573-583, ISSN 0169-8141,https://doi.org/10.1016/j.ergon.2011.05.001

Kılıçarslan, S., Adem, K., & Çelik, M. (2021). An overview of the activation functions used in deep learning algorithms. Journal of New Results in Science, 10(3), 75-88. https://doi.org/10.54187/jnrs.1011739

Hou, J., Wang, Y., Zhou, J., & Tian, Q. (2022). Prediction of hourly air temperature based on CNN–LSTM. Geomatics, Natural Hazards and Risk, 13(1), 1962–1986. https://doi.org/10.1080/19475705.2022.2102942