Cloud Coverage Detection

Made by: Kanishk Jain, Vishrut Grover, Hemang Jain (Team KVH)

Introduction to the Problem

Clouds play a crucial part in climate forecasting. Nevertheless, projecting future cloud fractional cover, that is, the percentage of the sky covered by clouds, carries a significant degree of uncertainty. The task of creating trustworthy climate projections is made more difficult by this substantial uncertainty.

Our Task: To build predictive models for cloud coverage prediction after the next 15, 25 and 30 minutes, given the data of past 360 minutes (6 hours)

Approaches Used

We have implemented various multivariate time series forecasting models like Simple RNNs, GRUs, LSTMs, Custom Transformer, Autoencoder, custom LSTM and a Conv1D + LSTM, to predict the cloud cover percentage from the numerical data. The training has been done on input sequences of 90 minutes (1.5 hours) and then predictions are taken for the next 15th, 25th and 30th minute.

1) Data Preprocessing

- Handling date and time in time series data, especially when the dates are in non-standard formats like **01-Jan** or **Jan-01**, requires changing the dates into consistent datetime format which makes them suitable for analysis.
- We scaled each feature individually to maintain the relative importance and variance of each feature. This is indeed crucial for models to learn meaningful patterns without being biased by features with larger ranges.
- After analysing all the features, **Snow Depth** and **Albedo (CMP11)** seem irrelevant so they were dropped.
- We did some feature engineering by combining **Tower Dry Bulb Temp [deg C]** and **Tower Dew Point Temp [deg C]** forming a new feature known as **Temperature-Humidity Index (THI)**.
- We also created **Potential Temperature** by combining **Tower Dry Bulb Temp [deg C]** and **Station Pressure [mBar]**
- We dropped **Peak Wind Speed @ 6ft [m/s]** and **Avg Wind Direction @ 6ft [deg from N]** and added **wind's X** and **Y** components from these.
- Finally, we dropped **Azimuth Angle [degrees]** and added its **Sine** and **Cosine**.

2) Model Implementations

Model Parameters:

- **n_past**: Number of time steps in each input sequence.
- **n_features**: Number of features at each time step.
- **future_pred**: Number of units in the output layer (e.g., predicting 3 features).

1) Simple RNN

- **Simple RNN Layer**: The input shape is specified here.
- **Dropout**: Helps prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.
- **Dense Layer**: Maps the final RNN output to the desired output shape.

2) GRU

- **GRU Layers with return_sequences**: These layers output the entire sequence of results for each input sequence, enabling the next GRU layer to handle the sequence one step at a time.

3) LSTM

- **LSTM Layers with return_sequences**: These layers return the full sequence of outputs for each input sequence, allowing the next LSTM layer to process the sequence step-by-step.

4) LSTM with concat layers

- LSTM layers process the input data. The outputs of these two LSTM layers are concatenated which effectively combines the features learned by both LSTM layers, doubling the number of features in the concatenated output.

5) Autoencoder

- An Autoencoder is a type of neural network used to learn efficient codings of input data. The **encoder** compresses the input data into low-dimensional space containing essential features, and the **decoder** reconstructs the output from this representation. The primary goal is to reconstruct the input itself.
- The **RepeatVector** layer takes a single vector and duplicates it multiple times. This transforms a 2D array (with dimensions batch_size by n_features) into a 3D array (batch_size, n_past, n_features), where n is the number of times the vector is repeated.
- The **RepeatVector** layer allows the model to handle cases where the input sequence length is different from the output sequence length. This is particularly useful for **forecasting future time steps** where the number of future steps (future_pred) is different from the past steps (n_past).

- By repeating the context vector (output of the first LSTM), the model can better learn the temporal patterns required to generate the output sequence. This helps in capturing dependencies over the repeated time steps.
- Using **TimeDistributed** is crucial when you want to apply a layer (like Dense) to each timestep of a sequence independently. It maintains the **temporal structure** of the data, ensuring that each timestep is processed in the context of its position within the sequence

6) Transformers

Transformer Encoder Blocks: (2 blocks)

1. **Layer Normalization**: Epsilon: 1e-6
2. **Multi-Head Attention**: Key Dim: 256, Heads: 4, Dropout: 0.25
3. **Dropout**: Rate: 0.25
4. **Layer Normalization**: Epsilon: 1e-6
5. **Residual Connection**: Input + Output of Normalization
6. **Feed Forward Network**:
    - Conv1D: Filters: 4, Kernel Size: 1, Activation: ReLU
    - Dropout: Rate: 0.25
    - Dense: Units: Input Shape Last Dimension
    - Layer Normalization: Epsilon: 1e-6
    - Residual Connection: Input + Output of Second Normalization

Positional Embedding: Embedding: Input Dim: 90, Output Dim: 15

Global Average Pooling: Data Format: "channels_last"

MLP (Multi-Layer Perceptron):

- Dense: Units: 128, Activation: ReLU, Dropout: Rate: 0.3
- Output Layer: Dense: Units: 3

7) Conv1D + LSTM layers

- **Conv1D** Layer: Convolutional layers are excellent at extracting local patterns or short term trends in the data. In time series forecasting models, the use of **Conv1D** and **MaxPooling1D** layers can help capture local structures while reducing input size before feeding it into **LSTM** cells. This method combines advantages of both recurrent and convolutional approaches;
- **LSTM** Layers: The **LSTM** layers process the input sequence to capture complex temporal patterns and long-term dependencies.

- Increasing the number of dense layers increases our accuracy as it can effectively combine the features learned by previous layers .

3) Model Compilation:

- **Optimizer**: Adam optimizer is used for efficient training.
- **Loss Function**: Mean Squared Error is used as the loss function for regression tasks.
- **Metrics**: We have used Mean Absolute Error, Mean Squared Error, R2 Score, and Root Mean Squared Error to evaluate the performance of your model during training.
- **Callbacks**: Early Stopping stops training when the validation loss stops to improve and Model Checkpoint saves the model at every epoch and ensures the best model is saved.

4) Final Predictions:

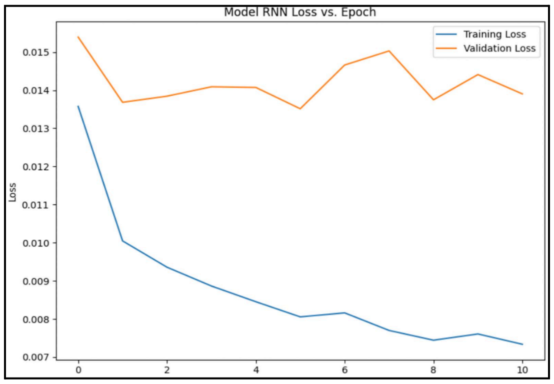
- To calculate the final accuracy metrics of our trained models, we used an averaging function that takes the mean of all the model accuracies.

Various Blockers

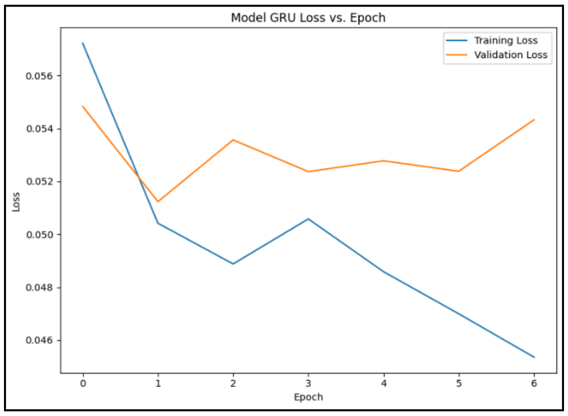
- Errors while handling date and time values.
- Long training time: With such a large dataframe, even a model with a small number of parameters takes a long time to train.
- Encountered unsatisfactory outcomes after removing NaN values from the dataset, so resorted to using the **interpolation** technique instead.
- Errors while converting **non-standard formats** like 01-Jan or Jan-01 into consistent datetime format that further be used for analysis.
- Had to separately convert the 29-Feb date manually.
- Errors while implementing Time Distributed Layer as it returns a 3D layer and loss function requires 2D data so flattened the output and returned.
- Transformer was harder to implement and was not giving the appropriate results as initially expected. This may have been due to the complex and longer architecture which may have led to some vanishing gradient problem during backpropagation.
- We attempted to use stacking and voting regressors for our final predictions, but encountered an error stating that they require data with less than 2 dimensions. Flattening the data did not resolve the issue either.

Visualizations

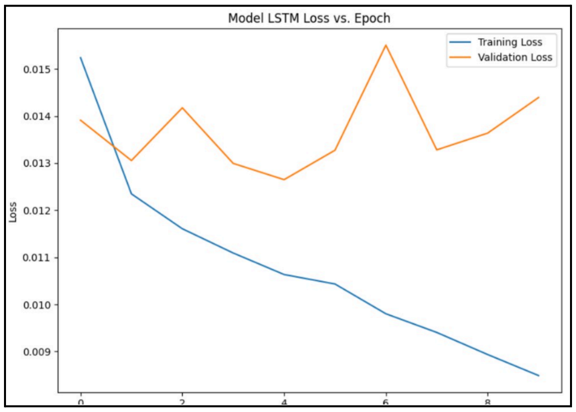
1) Simple RNN



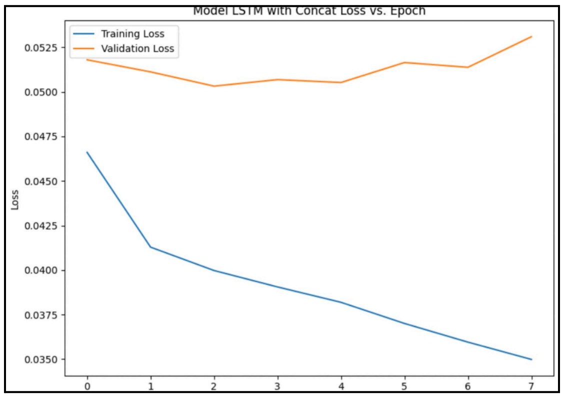
2) GRU



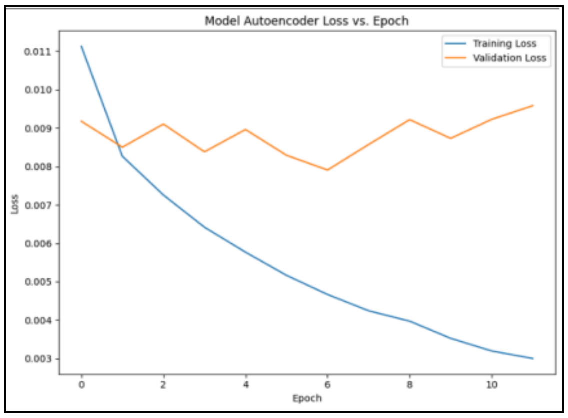
3) LSTM



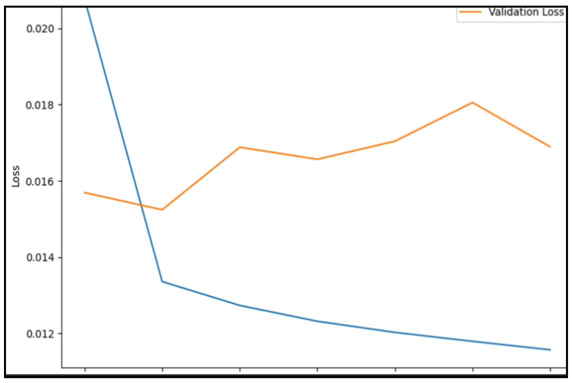
4) LSTM with concat



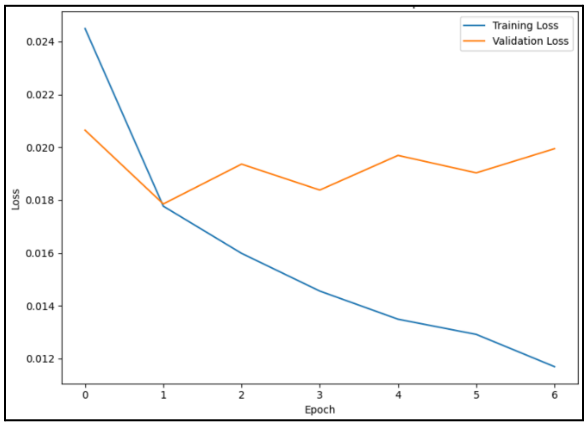
5) Autoencoder



6) Transformers



7) Conv1D + LSTM



Final Results

Weighted Metric = 0.5 x (metric for 15 mins) + 0.35 x (metric for 25 mins) + 0.15 x (metric for 30 mins) The metric here will be MSE, MAE and R2 score. We have also calculated the results for separate predictions.

Models	No. of Parameters	Weighted MAE	Weighted R2 Score	Weighted MSE
Simple RNN	11,251	3.710	0.900	70.3
LSTM	147,715	4.055	0.902	69.084
GRU	135,299	3.810	0.902	69.792
LSTM+Concat	178,467	3.692	0.899	71.627
Autoencoder	272,897	4.086	0.903	68.33
Custom Transformer	174,187	4.90	0.883	83.02

Conclusion and Future Prospects

- In conclusion, time series forecasting for weather data has made significant strides with the integration of machine learning techniques such as LSTM, CNN, and Transformer models. These approaches have improved the accuracy and reliability of weather predictions by capturing complex temporal patterns and dependencies.
- Transformer networks excel in capturing long-range dependencies but require substantial research to optimize performance and address issues like computational inefficiency and high memory usage. Combining it with complex LSTM Models and Ensembling can result in better networks for prediction, making Transformers a promising area for ongoing research and development.