Temperature Prediction Using Neural Networks

John Ludeman
Department of computer science
University of Nebraska
Omaha, Nebraska
jludeman@unomaha.edu

Austin Vornhagen
Department of computer science
University of Nebraska
Omaha, Nebraska
austinvornhagen@unomaha.edu

Krishna Priya Mooraka
Department of computer science
University of Nebraska
Omaha, Nebraska
kmooraka@unomaha.edu

ABSTRACT

In some areas of the US, internet and cell service are unavailable, and satellite connection is questionable. In these locations, extreme weather may catch people by surprise. If people in these regions have a chance to know weather conditions in advance, preparations can be made. Weather prediction is performed primarily using two methods: Numerical Weather Prediction (NWP) and Deep Learning Weather Prediction (DLWP). Previous research has been done on weather prediction by using NWP, however, other studies show that DLWP can outperform NWP. The purpose of this project will be to design a neural network that can predict the maximum and minimum temperature for the day on a single board computer, using time-series forecasting. The approach will be able to predict the temperature based on historical and current data. Similar approaches include bidirectional, long short-term memory (BiLSTM), convolutional neural network (CNN), transductive, long short-term memory (TLSTM), and many others each with their limitations. Even though using a DLWP approach is challenging on low-cost, resource-constrained hardware, it can give semi-accurate results. This neural network will run on a small single-board computer to make temperature predictions. The original contribution of this paper was to create a neural network light enough to do weather forecasting on a single-board computer.

Keywords—NWP, LSTM, Neural networks, DLWP, Single board computer.

I. INTRODUCTION.

Weather occurs in the lower atmosphere known as the troposphere which extends a few miles above the Earth’s surface. Weather forecasting is important in everyday life because people live in this troposphere. Weather influences most forms of transportation and the supply of food provided by farmers. Having an accurate weather forecast allows people to schedule their day and their travel plans in a safe way. Weather forecasting is also important for power grid prediction. For example, if the day is going to be abnormally hot, power usage increases significantly. If the grid operators are given notice in advance of upcoming weather conditions, the operators can perform more mitigation actions to protect the grid. Weather forecasts help home gardeners to protect plants from freezing weather. Prediction of temperature is a vital part of an individual’s decision of what clothing to wear. Weather prediction plays a key role in many aspects of civilization because the utility is very high. Weather prediction is performed primarily with two methods: Numerical Weather Prediction (NWP) and Deep Learning Weather Prediction (DLWP). In both methods, meteorologists gather data about several things that affect the weather like the wind, temperature, rain, UV light, humidity, solar irradiance, atmospheric pressure, etc. In NWP, statistical models are used to make predictions. In DLWP, the data is used to train neural networks that will then make predictions. Most current systems in use are designed for large-scale real-time monitoring and forecasting of weather conditions across an entire state or entire nation.

A. Artificial Neural Networks

A neural network is a type of program that can train on pre-labeled data, in this case, historical weather information. Based on the input information, the neural network will change the parameters or weights in each layer to recognize patterns in the data. The input data is the daily historical weather data going back several decades. The output will be the program's prediction for the max/min temperature for the day. Neural networks follow garbage in, garbage out policy. The quality of the inputs for the weather forecasting model will determine the quality of the outputs. The neural network architecture will also change the accuracy of the predictions.

B. Reccurent Neural Networks (RCNN)

Recurrent neural networks have a loop within each network, reoperating on previous outputs. This looping mechanism forms a highway-like flow of information from one step to another. The chain-like architecture allows it to recognize sequential characteristics in data and can predict the next step. Due to the advantage of its architecture, it can be used in various fields like speech recognition, image captioning, language modeling, translation, and many others.

C. Long Short Term Memory (LSTM)

Long short-term memory is a special recurrent neural network overcoming traditional recurrent neural network's long-term dependency problem. Instead of having a single layer in a repeating module, LSTMs contain four interacting layers, which solves a few of the problems with RCNNs.

D. Time Series Forecasting

Time series forecasting is the complex task of predicting future values of a given sequence using historical data [6]. This data has strict ordering. Forecasting can be done numerically with statistical methods or can be done by trying to find patterns with a neural network. In this paper, the weather data is structured as a time series with steps of one day.

E. Intel Galileo

The Intel Galileo is a model of single board computer (SBS). SBCs are complete computers built on a single circuit board. This includes a microprocessor, memory, input/output, and other features for a functional computer. An SBC can run an operating system, which will be conducive to this project.

F. Constraints

This project will be constrained by the abilities of the single-board computer (SBC). The SBC has limited memory and computational resources and we must be wary not to overextend the hardware. Another constraint is the availability of machine learning packages. Most of the powerful machine learning packages are written in Python (with many of the functions being written in C). The issue occurs when trying to run Python on Galileo board. To resolve this issue, an algorithm called Keras2c [18] was used to transfer the network into C. There are some implementations of machine learning in C and many more in C++, but the ease of use and power of the Python packages are impossible to beat.

.

II. Background and Related Work

This section reviews the literature on weather forecasting, time-series forecasting, deep learning, and embedded systems. This project will include using the Intel Galileo SBC to run a neural network that can forecast the weather. The problem identified is that accurate weather prediction is critical to daily life. Some other neural networks and systems exist, however, none of them run the forecast on the SBC.

Weather forecasting with a neural network has been studied [1], [2]. In these papers, weather forecasting techniques were analyzed with deep learning and artificial neural networks. The recommendation was that deep learning or an artificial neural network with backpropagation be used for weather forecasting. The main reasoning is that the neural network calculates the error and then tries to minimize the error. So low error would equate to a more accurate forecast. This is considered better than numerical weather prediction.

In [1], [2], the advice is very generalized, and the papers lack a detailed example of a practical neural network for weather forecasting. The majority is an analysis of and justification for using deep learning or an artificial neural network for weather forecasting in general.

Time-series forecasting using neural networks has been proposed in the energy sector and for weather forecasting [3], [4]. In these papers, a temporal convolutional neural network model (TCNN) is used to achieve accurate forecasts that could be executed quickly. TCNN is compared to the traditional LSTM model and the TCNN outperformed the LSTM in accuracy and predicting speed.

In [3], 6 Raspberry Pi weather stations gathered data with attached sensors to measure the atmosphere. This included a global system for mobile modules that allowed the weather board to send weather data to the server using the internet.

In [4], the accuracy of the model was evaluated with the weighted absolute percentage error. This paper shows that there are alternatives to the LSTM for time-series forecasting and a TCNN model should be considered as an alternative.

Temperature prediction with transductive LSTM was utilized. This is done by altering the cost function of the LSTM model [5]. It was found that transductive LSTM models can sometimes outperform the standard LSTM models. There is not enough evidence of the transductive LSTM outperforming the standard LSTM so it will not be considered in the new model design.

Time-series forecasting with a deep LSTM network was proposed in petroleum time series applications [6]. The deep LSTM is a standard LSTM but with more LSTM layers stacked. It was concluded that the deep LSTM could outperform a deep recurrent neural network, a deep gated recurrent unit, and the statistical ARIMA model when predicting a single variable. There was not any multi-variate time series forecasting performed.

COVID-19 transmissions were forecasted using an LSTM network [7]. The model predicted that the pandemic in Canada would last until June 2020. However, because of unreported cases, there could be infections for as long as December 2020. In present day (May 2022), Canada still has covid cases. This example shows either that this was a poor model, the situation was hard to predict, or the input data was not sufficient to predict the COVID-19 transmissions with real accuracy. Many factors can affect how a real outcome differs from a prediction.

Temperature forecasts have been explored and improved upon [8]. In this paper, a convolutional LSTM neural network was proposed. This network was outperformed by simpler models in the first few hours. Over larger forecasting horizons, meaning more than 6 hours, the convolutional LSTM neural network outperforms other methods. This kind of network could be considered as an alternative to a standard LSTM when doing weather forecasting.

Time-series forecasting using a WaveNet model has been proposed in the financial sector [9]. In this paper, a WaveNet model is used to do time series forecasting and compare it with the LSTM model and linear autoregressive model. The results did not find that the WaveNet model outperformed the LSTM model and linear autoregressive model, but WaveNet could compete. However, the WaveNet model appears prone to issues either with the ability to learn non-linearities or overfitting.

Others have tried to make a system that can forecast or track the weather with a small SBC or SBM without the internet [10], [11]. In [10], a cube-sat was designed and implemented which was based on a weather monitoring system with an Arduino Uno and a gas balloon. The system has 3 sensors: one for temperature and humidity, one for atmospheric pressure, and one sensor for altitude. 4 months of data were recorded with the device in Kolkata, India. The forecasts were generated with an Arduino.

In [10], there was not much information about the forecasting mechanism, just that the Arduino Uno received data from the sensors and performed the analysis. This design relies on a form of a weather balloon that would only work in calm weather.

In [11], the system only monitored the weather and output a text file of the data. It does not do any forecasting. This was made for implementation in a small area and to gather data for personal analysis. The paper was about designing, building, and successfully testing the weather monitoring system.

The Long-Short Term Memory layer [12] is a form of recurrent neural network. The major benefits of LSTM are the feedback connections and the sequential data processing. LSTM models contain four interacting layers: a cell, an input gate, an output gate, and a forget gate. When we train artificial neural networks with optimizers like stochastic gradient descent, a vanishing gradient can occur, which is the diminishing change of weights as the gradient becomes smaller. These additional layers in an LSTM module address the vanishing gradient.

Since the gates control the information into the cell, LSTM architectures can hold long-term dependencies. The first operation is a sigmoid function, which either keeps or disregards the input. The second operation is a combination of sigmoid and hyperbolic tangent, which is the decision of what to keep in the cell. This operation is carried out by the input layer which decides the values to update and creates a vector of the new candidate. A new cell state is then created based on these values. The output gate is a sigmoid layer that discriminates which parts of the cell are included in the input and a tanh which scales the output to [-1,1] from [0,1]. This operation keeps only the wanted parts of the cell state.

The paper [13] covers a design of temporal convolutional networks. These networks are designed so that no information leaks from the future into the past. Like a recurrent neural network, the temporal convolutional network takes a sequence of length n and maps it to an output sequence of length n. The convolutions are causal meaning an output can only depend on the elements that came before it. The model also used dilated convolution, where dilation is equivalent to introducing a gap between every application of the filter, allowing the neural network to look back further than just the span of the convolution mask.

[14] is referenced for understanding convolutional networks. In a convolutional network, each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer [17]. With local receptive fields, the nodes of the network can learn to extract elementary features.

Some conclusions can be drawn from the related work. There does not appear to be an agreed upon definition of the mandatory inputs for a weather prediction model. Example:

1) Temp, humidity, pressure
2) Temp, humidity, pressure, altitude
3) Temp, humidity, pressure, altitude, light
4) Etc.

The other papers made it clear that time-series forecasting is a complicated task and solving all time-series forecasting problems with one model is exceedingly difficult. It does not appear that anyone has run the neural networks used for weather forecasting on a single-board computer. Either the board sends data to a computer for neural network forecasting, the forecasting is done numerically or statistically, or the board only monitors the weather and does not forecast.

III. Methodologies/Framework/Architechture

This part of the paper presents the system of methods used in this study. The reader can evaluate the reliability and validity of the research.

A. The Architecture

In this section the architecture of Galileo board is explained in detail:

Fig.1 shows an image of the Galileo board, which is the hardware that will be performing the forecasts. It does not have an inbuilt Wi-Fi module so there will not be a way to connect to the internet with the board without plugging it into a computer. The data will have to be loaded onto the board manually. The processor is small, operating at speeds up to 400 MHz and the board only has 256 MB of RAM [15]. This is several times slower than the current standard processor and the reason it is challenging to run a neural network on the board successfully.

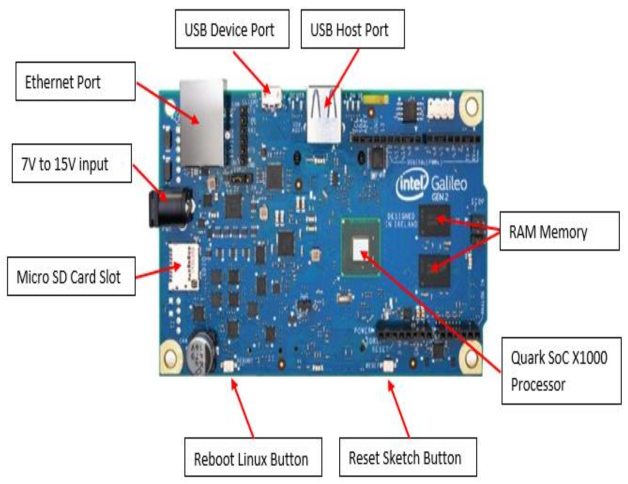


Fig. 1. Galileo board with components labeled. For more details, the reader is referred to reference [16] (part B. Intel Galileo Board Components [4] [5]).

B. Neural Network Models

This section briefly explains several types of neural network models:
Weather forecasting with machine learning, or more specifically a neural network, can be done and has been done in many ways. In the research, the following models were analyzed: LSTM model, deep LSTM model, WaveNet model, temporal convolutional model, transductive LSTM, deep recurrent neural network, deep gated recurrent unit, statistical autoregressive integrated moving average model, convolutional LSTM, linear autoregressive model. Here is a ranking of the best performing models for weather prediction based on the related work section:

1) Temporal Convolutional Model
2) Convolutional LSTM model
3) LSTM model
4) Transductive LSTM, deep LSTM model
5) Deep recurrent neural network, a deep gated recurrent unit, statistical ARIMA model, linear autoregressive model
6) WaveNet model

From the ranking, the Temporal Convolutional Model, Convolution LSTM model, and standard LSTM models performed the best for time-series forecasting and weather prediction. When choosing a model for the Galileo board, it is necessary to think about the limited computational resources available on a single board computer, the ability to transfer the python code into C so that it can run on the board, and the user friendliness of the model. The convolution LSTM (CLSTM) model and the standard LSTM model gave the highest probability of success when trying to forecast the weather on a Galileo board.

C. Neural Network Modeling Inputs

In this section the various types of data that are collected and how it is trained as an input to the neural network is discussed:

We collected various types of daily weather data from the Omaha, Nebraska area namely the minimum temperature, maximum temperature, average temperature, feels like maximum and minimum, snow depth, wind gust, precipitation, windspeed, humidity, sea-level pressure, cloud coverage, visibility, and dew for a particular area for the past twenty years. We have ranked all these and selected the two data points with the strongest distinguishing features to keep the neural network minimal to fit on the Galileo (single board computer). The two selected categories were minimum temperature and maximum temperature. The neural network was then trained using these categories. The source of the data is Visual Crossing and weather.gov. The data is stored in a .csv file. There was consideration to replace the .csv file with functionality to scrape the weather data from the websites, put it into a pandas data frame, then feed it into the neural network. The board could then have a way to continually update the dataset on its own. However, the main topic of study is designing a functional neural network that can perform forecasts on the Galileo board and this extra feature would only aid successive runs multiple days in a row.

D. Hardware and Operating Systems

This section discusses the hardware and operating systems that are used in this project:
There will not be any modifications to the Galileo board. At one point it seemed reasonable to add extra sensors like a thermometer and barometer onto the board. With these sensors, functionality could be added to give the board the ability to independently update the dataset within the .csv file. After more thought, it was decided that these sensors would increase the complexity of the project and take away from the focus of successfully performing a multi-variate forecast with a neural network on the Galileo board.

The initial idea was to use a robust Linux image to run the Python code, but there were many issues caused by the hardware of the board. Almost all machine learning packages for Python are based on a package called TensorFlow, which is only available in 64-bit. The Galileo processor is 32-bit. The method finally settled on was transferring the entire model into C. There is an amazing package, outlined in detail in [18], called Keras2C that takes the model architecture and the layer weights, and creates C functions. These functions have bare-minimum dependencies, relying only on basic libraries (stdio, math, string).

In the process of getting the program to run on Galileo board, multiple Linux images and Xinu have been considered. Xinu was a possible option, but with no standard library in the compilation, lack of easy file transfer, and lack of persistent file system, Xinu was disregarded. The next OS looked at and the one used was called Yocto Linux. Yocto Linux is specifically made with SBCs in mind, but many problems occurred when trying to build and run the image on the Intel Galileo. The Intel Galileo reached the end-of-life 5 years ago, with almost all forum posts and support pages being deleted.

Yocto Linux uses a system called Bitbake to create the image and include libraries/code. Bitbake contains layers, which contain the kernel, board support packages, and user generated code. Intel had provided the kernel and board support package layers, but the most recent version of Bitbake included was from 2014. The oldest version long term supported Ubuntu (14.04) was still too recent to have been tested with that version of Bitbake, but did eventually work. An outdated version of Git (1.7.5) was also required to be built from source for the application of patches to cloned repositories. Git version 1.9.2 was too high for this version of Bitbake. New versions of Bitbake would not work with the layers that Intel provides.

The generated code of the neural network using Keras2c was also needed to be changed due to the reliance on a compiled static library. Only a few changes to the include statements were made before the code worked. There was no detriment to the accuracy moving from Python on a desktop to C on the board.

The CSV file used for the prediction is uploaded over the local network using an SSH client. In this project the client used was MobaXterm. An SD card is used for booting in order to keep a persistent file system.

E. Trained Model

Trained neural network model that has been used in this project is explained in detail in this section:
The two neural networks (LSTM, CLSTM) will need to be tested and optimized for the weather dataset. This will be accomplished by changing the architecture, the hyper-parameters, and the input format. Testing may show that many, low parameter layers are superior to few, high parameter layers. For example, an LSTM model includes the recursive LSTM layer followed by a dense layer. The dense layer translates the latent dimension output of the LSTM layer to the prediction. Additional LSTM layers may be added before the dense layer to increase the recursion and possibly accuracy. The hyperparameters, such as drop out, latent dimension size, number of nodes, will also change to increase performance.

The machine learning package Keras is the backbone of this project. Keras, which is built on top of TensorFlow, is a deep learning API that provides a Python interface. Keras provides the neural network layers used for the LSTM and Convolutional LSTM. The convolutional LSTM is an emergent layer made from other simpler layers (convolutional, etc.) so the combination will be written using Keras. All matrix operations and data manipulation use the NumPy package.

The first neural network that is tested is a sequential model. There are two layers: 1 LSTM layer with 50 nodes, relu activation, and input shape of (10,2), 2 Dense layers with 2 nodes. The optimizer used is adam. The loss function means squared error.

The function that calculates the distance between expected output and current output of the algorithm is referred to as the loss function. The loss value is calculated by using mean squared error (MSE) for all the tests except the test specifically for changing the loss function.

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

The validation loss is the same as the loss function which assesses the performance of the neural network model, but it is run on the validation data set instead of the training data set.

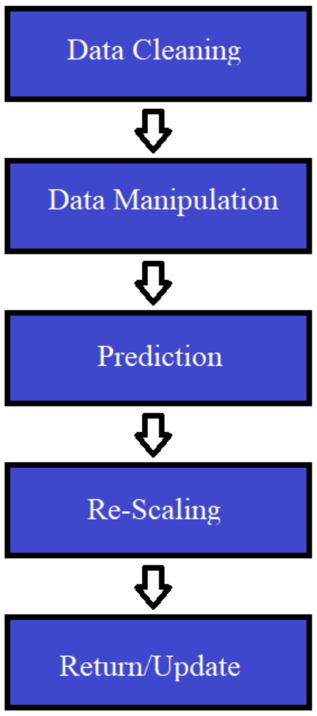


Fig. 2. Flow chart of final program

Fig.2 shows the flow chart of the final program.
In this section a detailed view of the internal process for the program is given:

1) Data cleaning will remove unwanted information and only grab the needed parameters from a specified .csv file. The parameters chosen are discussed more in-depth above.
2) Data Manipulation must occur in order to create a more accurate and faster training neural network. Since weights within a neural network are updated

based on loss, if the input and target data are scaled to be centered on 0 and between –1 and 1, the network will train faster and better.

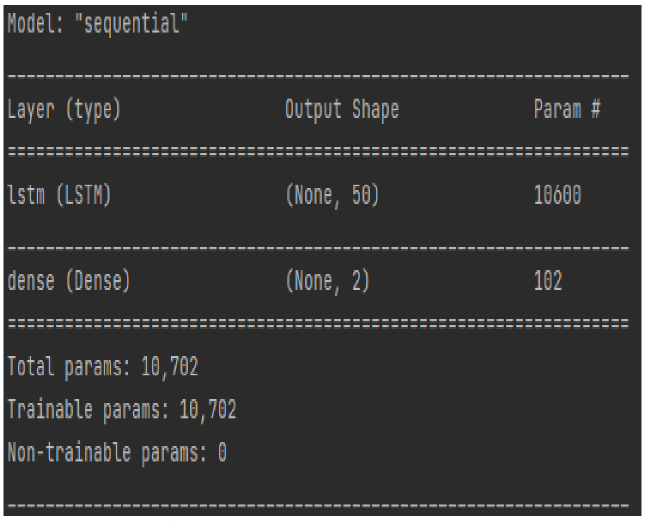
3) The prediction will then occur using a pre-trained neural network.
4) The output of the network must be unscaled using the same factors as the input to produce the true prediction values.
5) The program will then return the scaled prediction. Note: the data could be updated, and the network could be altered to perform several predictions in a row. This would allow the forecast to be more than a single time step. An example would be to generate a 7-day forecast. The current network only performs a single prediction.

Due to the hardware restrictions of the Galileo board, the neural network will never be trained once placed. The prediction is only one step, equivalent to one day. These models do not worry about long-term weather shifts and would require re-training and altering in order to do so.

IV. Experiments and Evaluation

The experiments and evaluation section covers the experiments that were carried out in order to validate that the neural networks performed forecasts accurately, to find the best model, and to make sure the neural network worked successfully on the Galileo board.

A. *Base Case – Default Models*



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                10600
_________________________________________________________________
dense (Dense)                (None, 2)                 102
=================================================================
Total params: 10,702
Trainable params: 10,702
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3. Summary of Standard LSTM model

Fig. 3 shows the layout of the standard LSTM model. The LSTM layer has output shape of 50 units, 10,600 parameters, and uses relu activation. The Dense layer has output shape of 2 units and 102 parameters. The model uses a lookback of 10 to predict the next step in the series. There are 2 input variables and 2 output variables tempmax and tempmin. The validation split is 0.2. The number of epochs used to train the model is 25. The batch size used in training the model is 32. The optimizer used is adam. The loss function used is mse (mean squared error). The loss value after 25 epochs is 0.0797. The validation loss value is 0.0844 after 25 epochs. The average difference between the predicted minimum temperature and the true minimum temperature was 4.13 degrees Fahrenheit. The average difference between the predicted maximum temperature and the true maximum temperature was 6.90 degrees Fahrenheit.

Fig.4 shows the model loss of the standard LSTM model. The loss of the training data and the loss of the validation data are stable. This would indicate that the model is fitting the data well without overfitting or underfitting. After roughly 5 epochs, the loss values remained stable for the rest of model training.

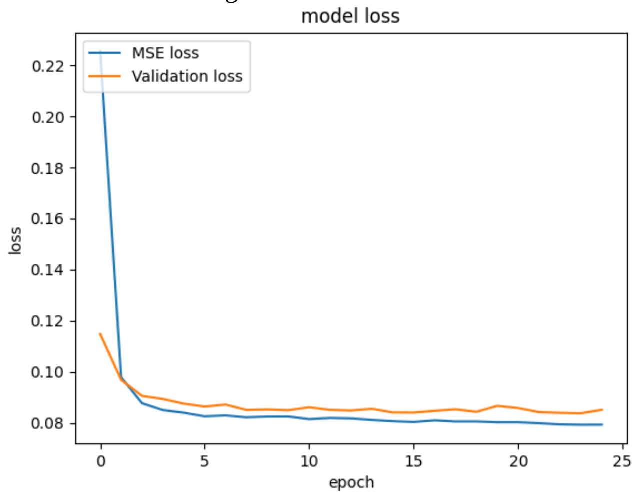


Fig. 4. Model Loss of Standard LSTM model

Fig. 5 and Fig. 6 show the minimum and maximum temperatures around Omaha that the model predicted vs the true temperatures in the data. Over the year of test data, the model is performing well in prediction.

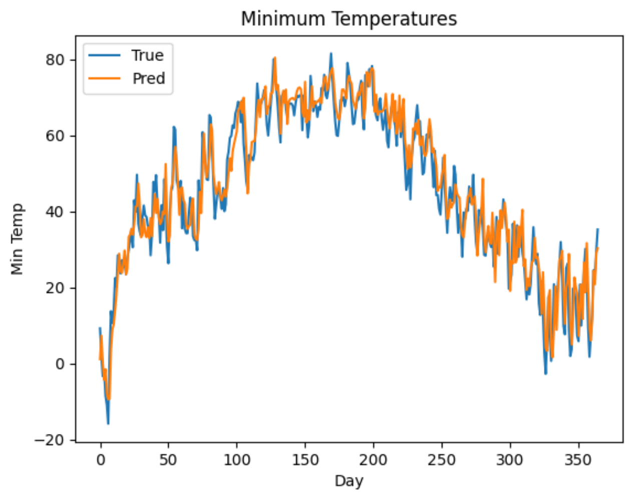


Fig. 5. Minimum Temperature Predictions of Standard LSTM model

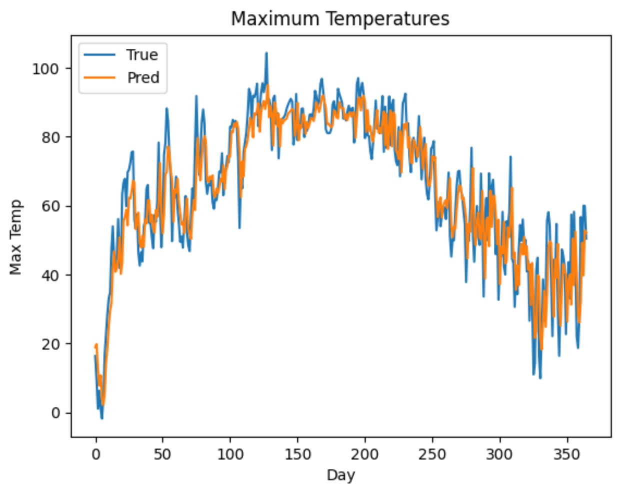


Fig. 6. Maximum Temperature Predictions of Standard LSTM model

Fig. 7 and Fig. 8 show the accuracy of the minimum and maximum temperature predictions as a percentage of the true value. The predictions have huge deviations from the true temperature values around the start and end of the test data. This is caused by the effect of errors on small data points. The test data is 365-days starting near the beginning of a year. When the temperatures are near zero, an error of 2 degrees can create a large percentage error. A prediction of 5 degrees when the true was 1 degree is quite accurate, but it is 500% of the true value causing a large spike in the graph. If this experiment were to be redone, the metric could be changed or adjusted to remove the large spikes in the data and make the graph more useful.

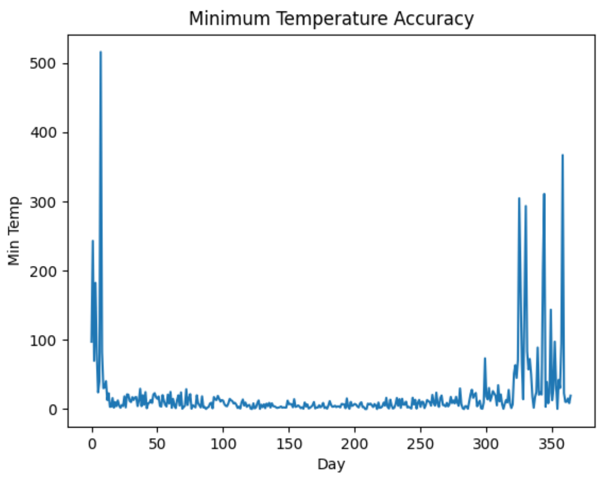


Fig. 7. Minimum Temperature Accuracy of Standard LSTM model

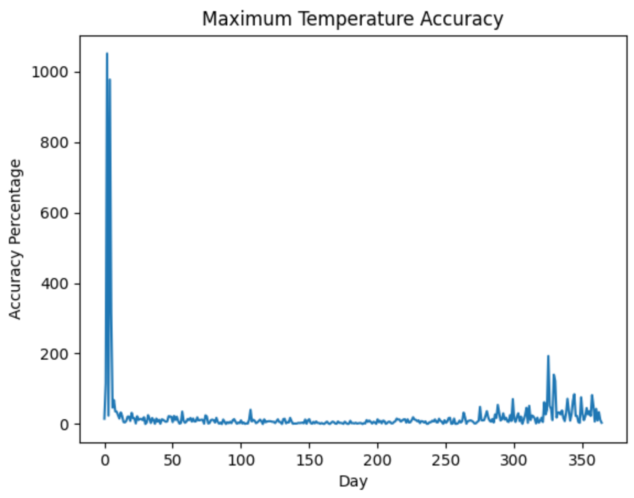


Fig. 8. Maximum Temperatures Accuracy of Standard LSTM model

Fig. 9 and Fig. 10 show the differences between the predicted minimum and maximum temperatures and the true temperature values. The prediction is sometimes above and sometimes below the true temperature value, so the graph shows the absolute value of the difference. The graph shows the model was off by about 17.5 degrees Fahrenheit on the high end of the minimum temperature and 25 degrees Fahrenheit on the high end of the maximum temperature. Sometimes the model had a perfect prediction on the low end.

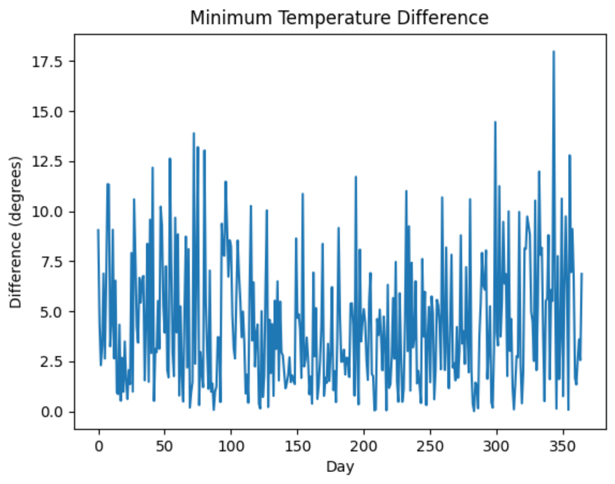


Fig. 9. Minimum Temperatures Differences of Standard LSTM model

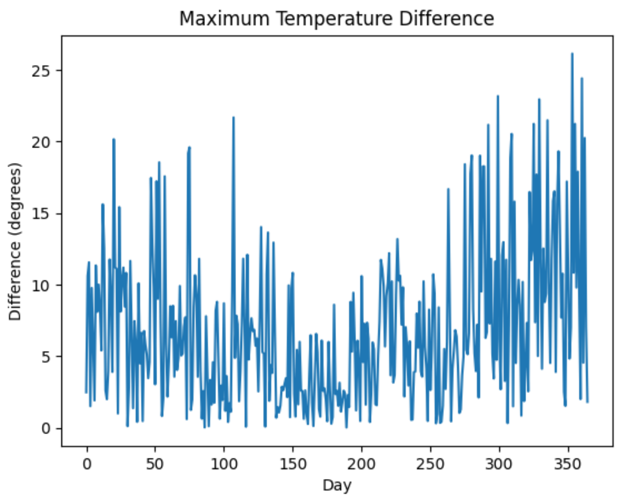
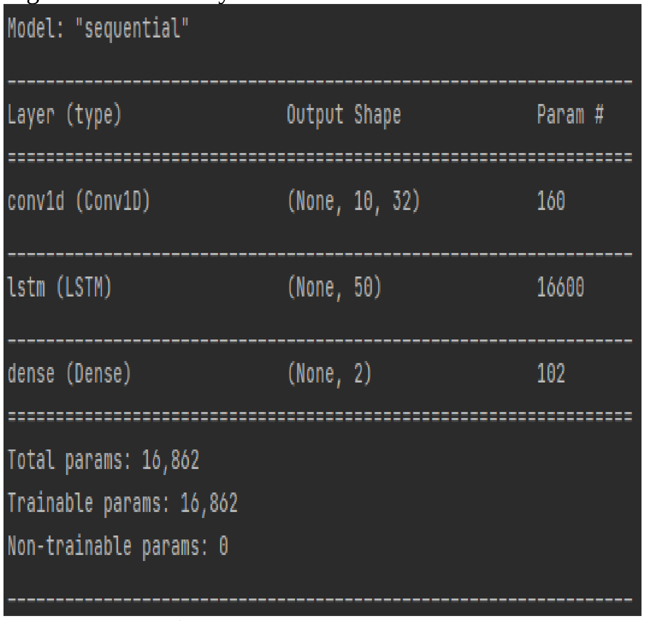


Fig. 10. Minimum Temperatures Predictions of Standard LSTM model

Fig. 11 shows the layout of the Convolutional LSTM model.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 10, 32)            160
_________________________________________________________________
lstm (LSTM)                  (None, 50)                16600
_________________________________________________________________
dense (Dense)                (None, 2)                 102
=================================================================
Total params: 16,862
Trainable params: 16,862
Non-trainable params: 0
_________________________________________________________________
```

Fig. 11. Summary of Convolutional LSTM model

The convolutional 1D layer has output shape of 10 vectors of 32-dimensional vectors, 160 parameters, and used relu activation. The LSTM layer has output shape of 50 units, 16600 parameters, and used relu activation. The Dense layer has output shape of 2 units and 102 parameters. The model uses a lookback of 10 to predict the next step in the series. There are 2 input variables and 2 output variables tempmax and tempmin. The validation split is 0.2. The number of epochs used to train the model is 25. The batch size used in training the model is 32. The optimizer used is adam. The loss function used is mse (mean squared error). The loss value after 25 epochs is 0.0770. The validation loss value is 0.0808 after 25 epochs. The average difference between the predicted minimum temperature and the true minimum temperature was 3.97 degrees Fahrenheit. The average difference between the predicted maximum temperature and the true maximum temperature was 6.74 degrees Fahrenheit.

Fig.12 shows the model loss of the Convolutional LSTM model. This shows the model picks up the pattern in the data very quickly after only about 5 epochs and is stable thereafter. The loss of the training data and the loss of the validation data are stable. This would indicate that the model is fitting the data well without overfitting or underfitting.

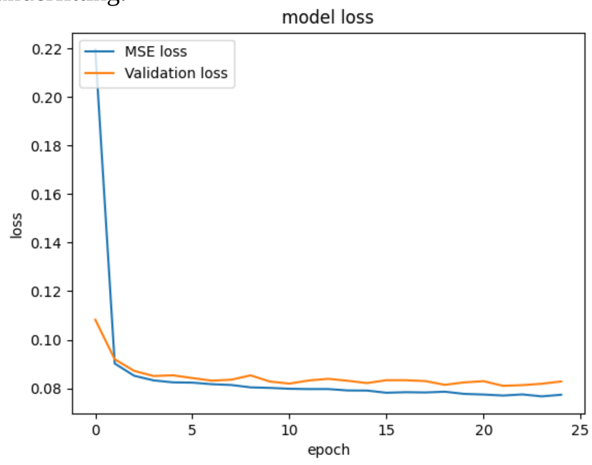


Fig. 12. Model Loss of Convolutional LSTM model

Fig. 13 and Fig. 14 show the minimum and maximum temperatures around Omaha the model predicted vs the true temperatures in the data. Over a year of test data, the model is performing well in prediction.

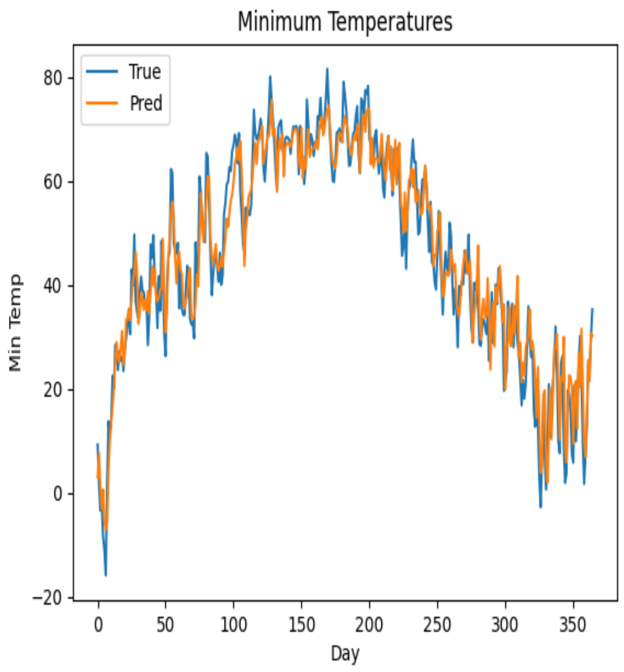


Fig. 13. Minimum Temperature Predictions of Convolutional LSTM model

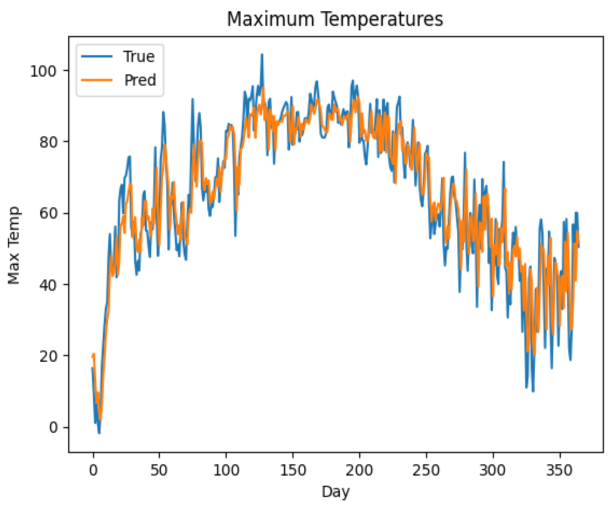


Fig. 14. Maximum Temperature Predictions of Convolutional LSTM model

Fig. 15 and Fig. 16 show the accuracy of the minimum and maximum temperature predictions as a percentage. The predictions have huge deviations from the true temperature values around the start and end of the test data. Like above, small magnitude errors on small underlying data can create the large percentage error.

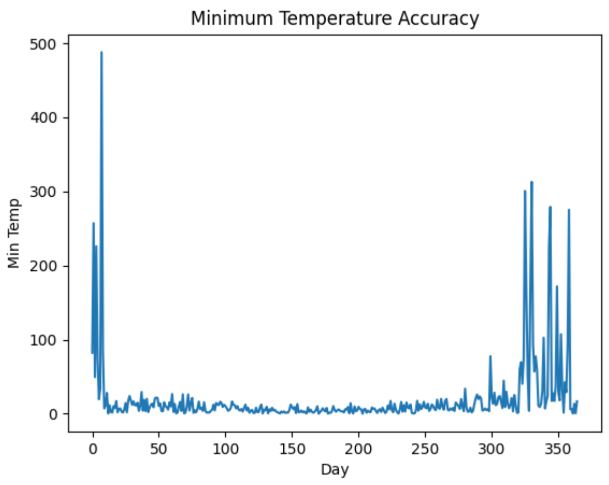


Fig. 15. Minimum Temperature Accuracy of Convolutional LSTM model

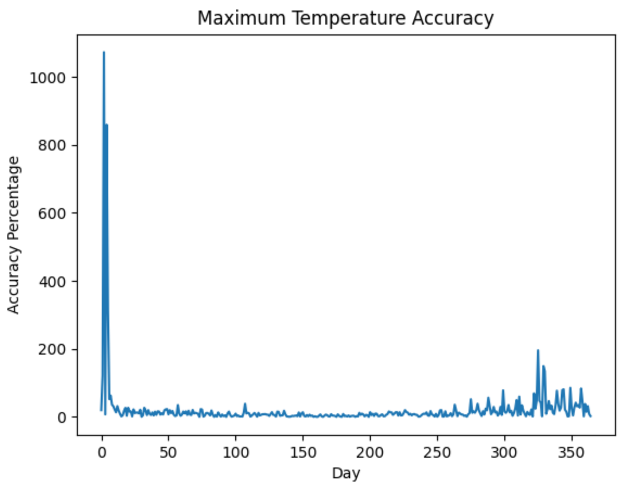


Fig. 16. Maximum Temperature Accuracy of Convolutional LSTM model

Fig. 17 and Fig. 18 show the differences between the predicted minimum and maximum temperature and the true temperature values. Like above, there was a range the model predictions oscillated within.

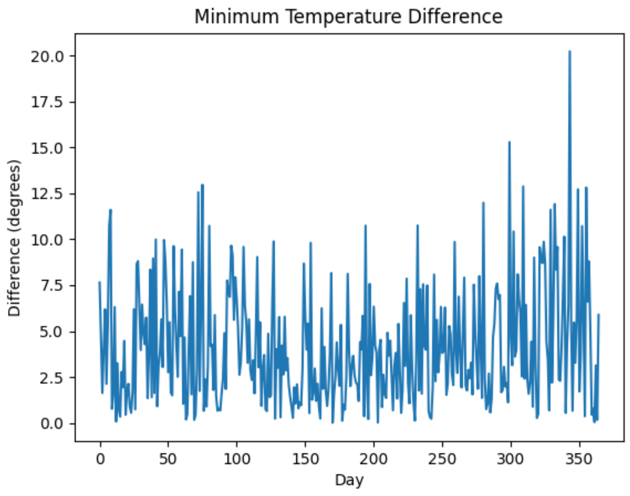


Fig. 17. Minimum Temperatures Differences of Convolutional LSTM model

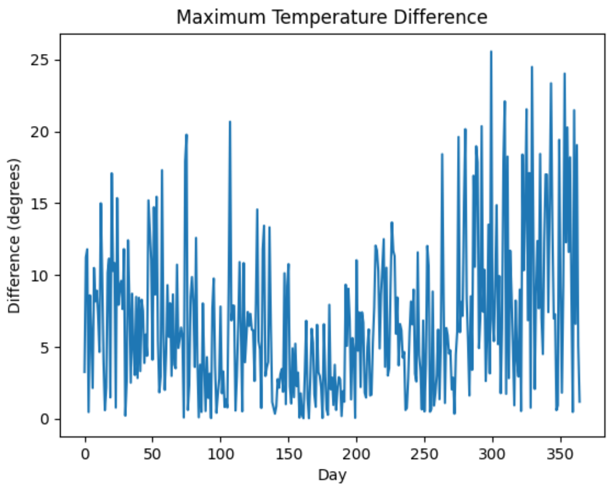


Fig. 18. Maximum Temperatures Differences of Convolutional LSTM model

B. *Experiment 1 – Adding Layers to Default Models*

Table 1 shows the standard LSTM model performance when extra LSTM layers are added. It performs slightly better than the base case when 1 extra LSTM layer is added to the model. When 2 extra LSTM layers are added the results are about the same as the base case. When 5 extra LSTM layers are added the loss value and validation loss both increases. The difference between the loss and validation loss is lowest with 5 extra LSTM layers added to the model. Some observations are that the run time of the model increases as more layers are added and the memory the model occupies also increases.

Standard LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
Base Case	0.0797	0.0844	0.0047	4.13	6.90
1 Extra LSTM Layer	0.0792	0.0818	0.0026	4.09	6.84
2 Extra LSTM Layers	0.0808	0.0852	0.0044	4.27	6.89
5 Extra LSTM Layers	0.0851	0.0860	0.0009	4.35	7.08

Table.1 The standard LSTM (experiment 1)

Table 2 shows the convolutional LSTM model performance when extra convolutional layers are added. It performs slightly worse than the base case when 1 extra convolutional layer is added. Both the loss value and validation loss values decreased. The difference in average temperature predictions increased for minimum and maximum temperature. When 2 extra convolutional layers are added the model performs continually worse than the base case. The loss value and validation loss value have increased and the difference between them has also increased. When 5 extra convolutional layers are added the model performs the worst. An observation is the difference between the loss value and validation loss value got worse each time another convolutional layer was added. This may suggest that the model fits that data less as more convolutional layers are added, and it would be better to only have a single layer like in the base case.

Convolutional LSTM	Loss Value	Validation Loss	Validation – Loss	Average Min Diff	Average Max Diff
Base Case	0.0770	0.0808	0.0038	3.97	6.74
1 Extra Conv Layer	0.0757	0.0807	.005	4.11	6.81
2 Extra Conv Layer	0.0767	0.0834	0.0067	4.18	6.91
5 Extra Conv Layer	0.0771	0.0865	0.0094	4.32	6.75

Table.2 The convolutional LSTM (experiment 1)

C. Experiment 2 – Changing Hyperparameters

Table 3 shows the standard LSTM model when the dimensions of the LSTM layer are adjusted. It appears the loss value improves with more dimensions. However, the validation loss value, the difference between loss and validation loss, and the average min and max temperature differences are mostly unaffected.

Standard LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
10 LSTM Dims	0.0806	0.0847	0.0041	4.24	6.99
30 LSTM Dims	0.0802	0.0856	0.0054	4.20	6.79
Base Case 50 LSTM Dims	0.0797	0.0844	0.0047	4.13	6.90
100 LSTM Dims	0.0790	0.0828	0.0038	4.04	6.84
200 LSTM Dims	0.0786	0.0839	0.0053	4.21	6.91

Table.3 The standard LSTM (experiment 2)

Table 4 shows the convolutional LSTM model when the filters in the convolutional layer are adjusted. It appears that the loss value improves with more filters. However, the base case with 32 filters seems to be the best performing setting for validation loss, the difference between loss and validation loss, and the average min and max differences.

Convolutional LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
10 filters	0.0798	0.0848	0.005	4.21	6.87
20 filters	0.0774	0.0823	0.0049	4.06	6.77
Base Case 32 filters	0.0770	0.0808	0.0038	3.97	6.74
100 filters	0.0763	0.0813	0.005	4.04	6.75
200 filters	0.0758	0.0866	0.0108	4.50	7.17

Table.4 The convolution LSTM (experiment 2)

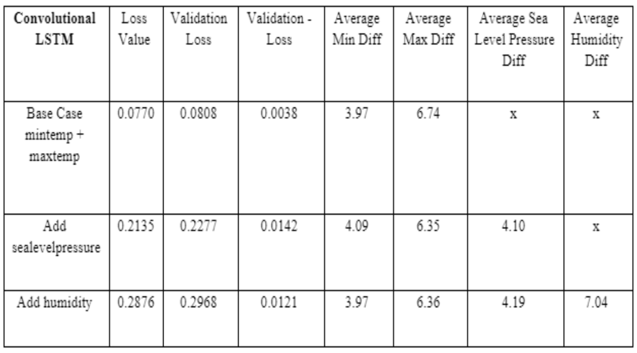
D. Experiment 3 – Adding Inputs

Table 5 shows the standard LSTM model when additional features are added. We can see clearly that more features lead to higher loss value and validation loss value. All other metrics do not have a definitive pattern.

Standard LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff	Average Sea Level Pressure Diff	Average Humidity Diff
Base Case mintemp + maxtemp	0.0797	0.0844	0.0047	4.13	6.90	x	x
Add sealevelpressure	0.2175	0.2240	0.0065	4.14	6.32	4.06	x
Add humidity	0.2948	0.2969	0.0021	3.96	6.41	4.09	6.83

Table.5 The standard LSTM (experiment 3)

Table 6 shows the convolutional LSTM model when additional features are added. Like the standard LSTM more features cause higher loss value and validation loss value. All other metrics do not have a definitive pattern.

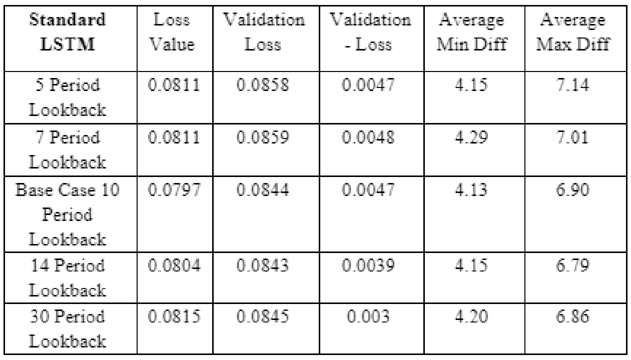


Convolutional LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff	Average Sea Level Pressure Diff	Average Humidity Diff
Base Case mintemp + maxtemp	0.0770	0.0808	0.0038	3.97	6.74	x	x
Add sealevelpressure	0.2135	0.2277	0.0142	4.09	6.35	4.10	x
Add humidity	0.2876	0.2968	0.0121	3.97	6.36	4.19	7.04

Table.6 The convolution LSTM (experiment 3)

E. Experiment 4 – Changing Lookback

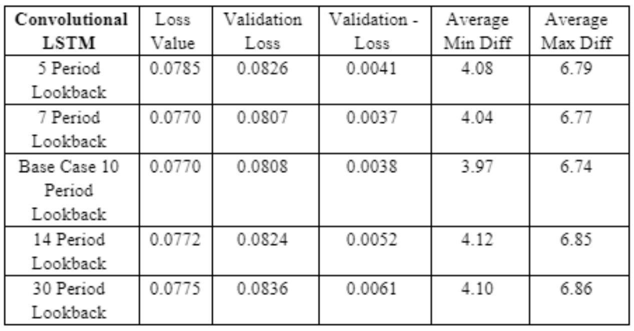
Table 7 shows the standard LSTM model when the lookback value is changed. All values are mostly unaffected by this change.



Standard LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
5 Period Lookback	0.0811	0.0858	0.0047	4.15	7.14
7 Period Lookback	0.0811	0.0859	0.0048	4.29	7.01
Base Case 10 Period Lookback	0.0797	0.0844	0.0047	4.13	6.90
14 Period Lookback	0.0804	0.0843	0.0039	4.15	6.79
30 Period Lookback	0.0815	0.0845	0.003	4.20	6.86

Table.7 The standard LSTM (experiment 4)

Table 8 shows the convolutional LSTM model when the lookback value is changed. All metrics are mostly unaffected by this change.

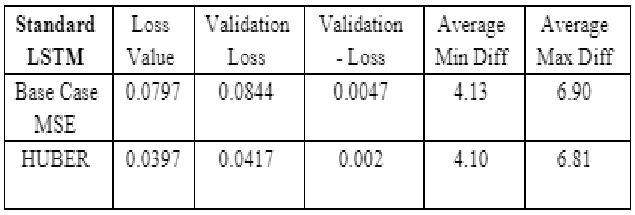


Convolutional LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
5 Period Lookback	0.0785	0.0826	0.0041	4.08	6.79
7 Period Lookback	0.0770	0.0807	0.0037	4.04	6.77
Base Case 10 Period Lookback	0.0770	0.0808	0.0038	3.97	6.74
14 Period Lookback	0.0772	0.0824	0.0052	4.12	6.85
30 Period Lookback	0.0775	0.0836	0.0061	4.10	6.86

Table.8 The convolution LSTM (experiment 4)

F. Experiment 5 – Changing Loss Function

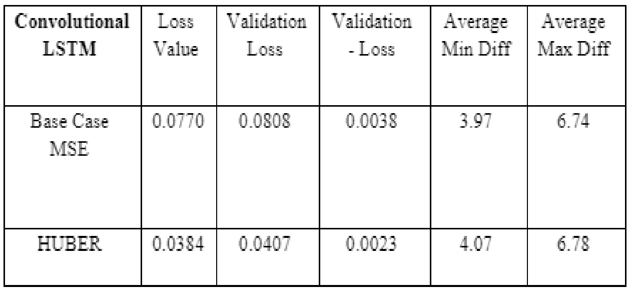
Table 9 shows the standard LSTM model when the loss function is changed to huber. Huber is used to decrease the amount of power outliers have in affecting the ending loss value. The huber function decreases all metrics and improves performance of the model in all measured areas. The huber loss function shows the model fits the data better than the base case shows.



Standard LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
Base Case MSE	0.0797	0.0844	0.0047	4.13	6.90
HUBER	0.0397	0.0417	0.002	4.10	6.81

Table.9 The standard LSTM (experiment 5)

Table 10 shows the convolutional LSTM model when the loss function is changed to huber. The huber function shows a much lower loss value, validation loss value, and difference between validation loss and loss. The average minimum and maximum temperature differences increased slightly when using the huber loss function.



Convolutional LSTM	Loss Value	Validation Loss	Validation - Loss	Average Min Diff	Average Max Diff
Base Case MSE	0.0770	0.0808	0.0038	3.97	6.74
HUBER	0.0384	0.0407	0.0023	4.07	6.78

Table.10 The convolution LSTM (experiment 5)

G. Evaluation

The approach of this project is to forecast the weather using DLWP. The data is formatted in a time-series which complements the DLWP approach. The set of experiments helps to get a full understanding of the current performance of the models and what optimizations can be made in order to increase performance. This is important for trying to make accurate forecasts.

One issue with this project was getting the neural network to run on the board. This requires transferring the networks into C. Another issue was getting the neural networks to read from the csv file on the Galileo board. The two models that were evaluated were similar and including both did not add much to the report. The plan was to add a 3rd temporal convolutional neural network model for comparison but there was not enough time to add it because it requires a separate machine learning Python package. This other package did not have a readily available C conversion making it near impossible to run on the Galileo.

V. CONCLUSION

In this paper, two deep learning models were compared and executed on an Intel Galileo Gen 2 single board computer in order to forecast the weather. The first model was a standard LSTM model and the second was a Convolutional LSTM model.

The results that a neural network can be light enough to run on a single board computer are significant because they prove that the power of deep learning can be removed from a computer and used on hardware that can be taken to remote places where technology may not otherwise be able to be used. The results also confirm that a standard LSTM and Convolutional LSTM are great networks for weather forecasting on a single board computer.

The Intel Galileo board was created several years ago, and these results are limited by the capabilities of the hardware. In the future, more research could be done on additional networks to see which would provide the most accurate forecasts. There are other packages that exist for doing machine learning on small devices with low power like tinyML and TensorFlow Lite [19], [20]. These could be explored for more efficient forecasting models. There are also other kinds of hardware that can be compared with the Intel Galileo board for better performance and updated capabilities. The field of applying deep learning to small low power devices is expanding and this research should remain relevant going forward.

VI. References

[1] Ren, X., Li, X., Ren, K., Song, J., Xu, Z., Deng, K., & Wang, X. (2021). Deep learning-based weather prediction: a survey. Big Data Research, 23, 100178.

[2] Narvekar, M., & Fargose, P. (2015). Daily weather forecasting using artificial neural network.

[3] Hewage, P., Behera, A., Trovati, M., Pereira, E., Ghahremani, M., Palmieri, F., & Liu, Y. (2020). Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station. Soft Computing, 24(21), 16453-16482.

[4] Lara-Benítez, P., Carranza-García, M., Luna-Romera, J. M., & Riquelme, J. C. (2020). Temporal convolutional networks applied to energy-related time series forecasting. applied sciences, 10(7), 2322.

[5] Karevan, Z., & Suykens, J. A. (2020). Transductive LSTM for time-series prediction: An application to weather forecasting. Neural Networks, 125, 1-9.

[6] Sagheer, A., & Kotb, M. (2019). Time series forecasting of petroleum production using deep LSTM recurrent networks. Neurocomputing, 323, 203-213.

[7] Chimmula, V. K. R., & Zhang, L. (2020). Time series forecasting of COVID-19 transmission in Canada using LSTM networks. Chaos, Solitons & Fractals, 135, 109864.

[8] Kreuzer, D., Munz, M., & Schlüter, S. (2020). Short-term temperature forecasts using a convolutional neural network—An application to different weather stations in Germany. Machine Learning with Applications, 2, 100007.

[9] Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691.ma

[10] Laskar, M. R., Bhattacharjee, R., Giri, M. S., & Bhattacharya, P. (2016). Weather forecasting using Arduino based cube-sat. Procedia Computer Science, 89, 320-323

[11] Krishnamurthi, K., Thapa, S., Kothari, L., & Prakash, A. (2015). Arduino based weather monitoring system. International Journal of Engineering Research and General Science, 3(2), 452-458.

[12] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[13] Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.

[14] LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object recognition with gradient-based learning. In Shape, contour and grouping in computer vision (pp. 319-345). Springer, Berlin, Heidelberg.

[15] https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/galileo-g2-datasheet.pdf

[16] https://www.academia.edu/27302409/A_Review_of_Intel_Galileo_Development_Board_s_Technology

[17] Lv, G. (2011, October). Recognition of multi-fontstyle characters based on convolutional neural network. In 2011 Fourth International Symposium on Computational Intelligence and Design (Vol. 2, pp. 223-225). IEEE.

[18] Colin, R., Erickson, K., Abbate, J., Koleman, E., Keras2c: A library for converting Keras neural networks to real-time compatible C

[19] https://www.tinyml.org

[20] https://www.tensorflow.org/lite