Weather Prediction using TinyML

Parsheeta Roy
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`parsheer@andrew.cmu.edu`

Shreya Ajay Kale
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`shreyaak@andrew.cmu.edu`

1 Abstract

This project aims to develop a cost-effective weather forecasting system using the Arduino Nano 33 BLE Sense. By utilizing Tiny Machine Learning (TinyML), our model can run efficiently on constrained hardware, enabling real-time weather predictions without internet connectivity. This system can be particularly relevant for users in resource-limited areas (including outdoor enthusiasts, remote weather stations and educational institutions), where access to continuous network connectivity or high-powered hardware is often unavailable or expensive.

Our approach aims to deliver reliable weather predictions by capturing key environmental metrics like temperature and barometric pressure. This will ensure that the system remains practical for low-resource applications, providing accurate and timely forecasts even in areas with minimal infrastructure. Our goal is to contribute to broader accessibility in cost-efficient weather forecasting, and our code is available at `https://github.com/parsheetaroy/Pittsburgh-Weather-Forecasting-using-Arduino-Nano-33-BLE-Sense` to encourage future reproducibility.

2 Dataset

We sourced daily Pittsburgh weather data from Weather Underground for the last 10 years, and manually preprocessed the historical records to ensure the data was clean, standardized, and well-structured for analysis.

Initially, we filtered the dataset to retain only relevant columns, addressing missing or invalid values by removing rows with null or erroneous entries. To enable seamless temporal analysis, we created a new `Datetime` column by combining the `Date` and `Time` fields, and the data was then sorted chronologically to maintain proper sequencing.

Temperature values were standardized by converting them from Fahrenheit to Celsius, while pressure readings were transformed from inches to hectopascals (hPa). To simplify analysis, we consolidated the original 63 weather condition classes into broader categories—*Cloudy*, *Sunny*, *Rainy*, *Snowy*, and *Other*—using a predefined mapping. Any unmapped or missing conditions were manually addressed to ensure consistency and accuracy.

The final dataset included columns for `Datetime`, standardized temperature, pressure, and consolidated weather conditions. These preprocessing steps ensured that the dataset was consistent, and well-prepared for subsequent weather classification and forecasting tasks.

Preprint. Under review.

3 Weather forecasting pipeline

This pipeline is designed to predict weather features, such as temperature, pressure, and broad weather conditions(Cloudy, Sunny, Rainy, Snowy, Other), using a simple feedforward neural network (FFNN). The following steps outline the entire process in detail:

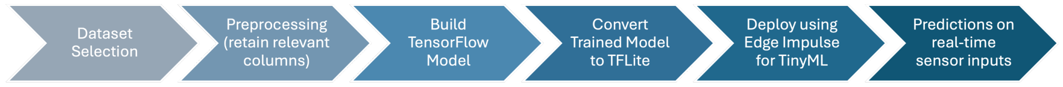
Dataset Selection
Preprocessing (retain relevant columns)
Build TensorFlow Model
Convert Trained Model to TFLite
Deploy using Edge Impulse for TinyML
Predictions on real-time sensor inputs

Figure 1: Overall Pipeline

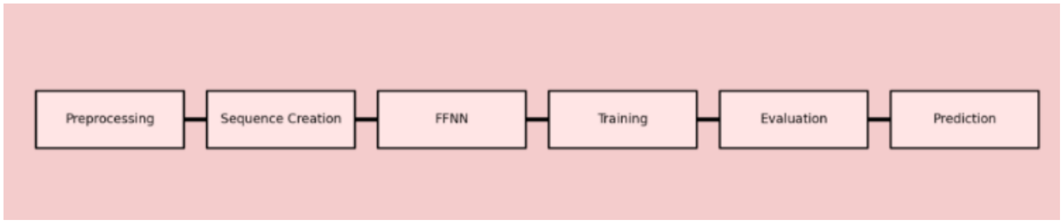

Figure 2: Deep Learning Pipeline for TinyML Weather forecasting and classification

3.1 Feature Selection and Extraction

The dataset from Section 2 is loaded as a CSV file containing historical weather records. The preprocessing phase begins by applying one-hot encoding to the categorical variable `Broad Condition` using `OneHotEncoder` from `scikit-learn`. This converts the categorical weather conditions into numerical vectors suitable for input to the model. Simultaneously, numerical features, including `Temperature (C)` and `Pressure (hPa)`, are normalized to the range $[0, 1]$ using Min-Max scaling. These steps ensure that all input features have comparable magnitudes, facilitating effective model training.

The processed numerical and categorical data are combined and structured into sequences for supervised learning. Each sequence spans 30 consecutive days (controlled by the `past_days` parameter), with the target output being the weather data for the subsequent day. The sequences are then split chronologically into training, validation, and testing datasets in a 70:10:20 ratio. This ensures that the temporal order of the data is preserved.

3.2 Model Architecture

The weather forecasting model is built using a Feedforward Neural Network (FFNN) implemented in `Keras`. The architecture is carefully designed to handle both numerical and categorical features, ensuring robust forecasting capabilities. The detailed architecture is as follows:

- **Input Layer:** The input layer accepts a flattened sequence of shape (None, 128), where 128 corresponds to the combined numerical and categorical features over a 30-day window.
- **Hidden Layers:**
  - A **Dense layer** with 128 neurons and a `ReLU` activation function is used to capture complex patterns in the data. This layer is followed by a **Dropout layer** with a 30% dropout rate to reduce overfitting.
  - Another **Dense layer** with 64 neurons and a `ReLU` activation function is added, followed by a **Dropout layer** with a 20% dropout rate for additional regularization.
  - A final **Dense layer** with 32 neurons and a `ReLU` activation function further refines the learned features.
- **Output Layer:** The output layer simultaneously predicts:
  - **Numerical Features:** Temperature and pressure, predicted using two nodes.

    - **Categorical Feature:** Weather condition, predicted using a softmax activation function over multiple classes (e.g., Cloudy, Rainy, Sunny, etc.).
- **Parameters:** The model consists of a total of 37,575 parameters, all of which are trainable.

The architecture ensures that both numerical and categorical outputs are handled effectively, with the final layer providing predictions that align with the multifaceted nature of the weather forecasting task.

3.3 Training and Checkpointing

The FFNN is compiled using the `Adam` optimizer with a Learning rate of 0.001 and trained on the processed dataset. During training, callbacks are employed to enhance performance and prevent overfitting. Specifically:

- `EarlyStopping` halts training if the validation loss does not improve for 5 consecutive epochs.
- `ReduceLROnPlateau` reduces the learning rate by a factor of 0.5 if validation loss stagnates, ensuring smoother convergence.
- `ModelCheckpoint` saves the best model based on validation loss.

3.4 Evaluation Metrics

After training, the FFNN is evaluated on the test set using the custom loss function. To interpret the model's performance, these metrics are calculated:

- **Test Loss:** The model achieves a test loss of 0.5634, indicating the combined error for numerical and categorical predictions.
- **Mean Absolute Error (MAE):** The model demonstrates an MAE of 2.94°C for temperature predictions and 4.95 hPa for pressure predictions.
- **Accuracy:** For the categorical `Broad Condition`, the model achieves an accuracy of 81.04%.
- **Predicted vs. Actual:** For the last sequence in the test set, the model predicted the next day's weather with the following results:
    - **Temperature:** Predicted 10.97°C vs. Actual 10.00°C.
    - **Pressure:** Predicted 974.37 hPa vs. Actual 969.86 hPa.
    - **Condition:** Predicted ['Cloudy'] vs. Actual ['Cloudy'].

These metrics indicate the model's overall performance in forecasting both numerical weather features and broad weather conditions, with reasonably low error rates and high classification accuracy.

4 Deployment on Arduino Nano 33 BLE Sense

The **Arduino Nano 33 BLE Sense** served as the primary hardware platform for this project, chosen for its compact size, low power consumption, and integrated suite of environmental sensors. After training our Feedforward Neural Network (FFNN) model, it was converted into the `TensorFlow Lite` (tflite) format to make it compatible with the resource-constrained microcontroller. The deployment process involved using **Edge Impulse** to build the TinyML project, enabling integration of the tflite model onto the Nano 33 BLE Sense.

Additionally, we utilized the **Arduino IDE** to develop a code that collected real-time data from the onboard **LPS22HB pressure sensor**. This sensor provided precise measurements of *barometric pressure* and ambient temperature, allowing us to capture and preprocess these two essential environmental metrics in real time.

The recorded temperature and pressure data were fed into the deployed tflite model on the Arduino Nano 33 BLE Sense, which generated predictions for the next day's weather conditions along with temperature and pressure values.

5 Significant Challenges and Lessons Learned

During the development of this project, we encountered several challenges that required thoughtful solutions and adaptations.

5.1 Dataset Acquisition

One of the primary challenges was acquiring a relevant weather dataset specific to Pittsburgh with sufficient time-series data and corresponding weather condition labels. Many publicly available datasets lacked granularity or structured weather condition labels, which was essential for training our forecasting model.

5.2 Model Complexity and Hardware Constraints

To achieve higher accuracy, we initially experimented with **RNN**, **LSTM** and **Bi-LSTM** models, given their effectiveness with time-series data. However, the computational constraints of the **Arduino Nano 33 BLE Sense** made these models impractical for deployment. We pivoted to a simpler **Feedforward Neural Network (FFNN)**, optimizing it to deliver comparable accuracy while being lightweight and efficient for the target hardware.

5.3 Lessons Learned

This project emphasized the importance of balancing model performance with hardware limitations. Simplified models, when well-optimized, can achieve reliable accuracy while being computationally efficient for edge deployment.

6 Demonstration video

Here is a small video demonstration of our project - `https://drive.google.com/file/d/1j_0qvJeLj3dffRLFPTZKns47gFfG7JmB/view?usp=sharing`