Instructions List :

* Setting up ESP32 board
* IoT long-range Wireless Temperature and Humidity sensor
* Making connections between the Wireless temperature sensor and ESP32
* Flashing ESP 32 and Collecting Sensor data from Wireless Temperature and Humidity sensor
* Analyzing and visualizing the sensor data in charts and graphs using Power BI
* Exporting the Sensor data in CSV from Power BI
* Creating a data set using this data.
* Creating a CNN(Convolutional Nueral Network Model) using tensorflow

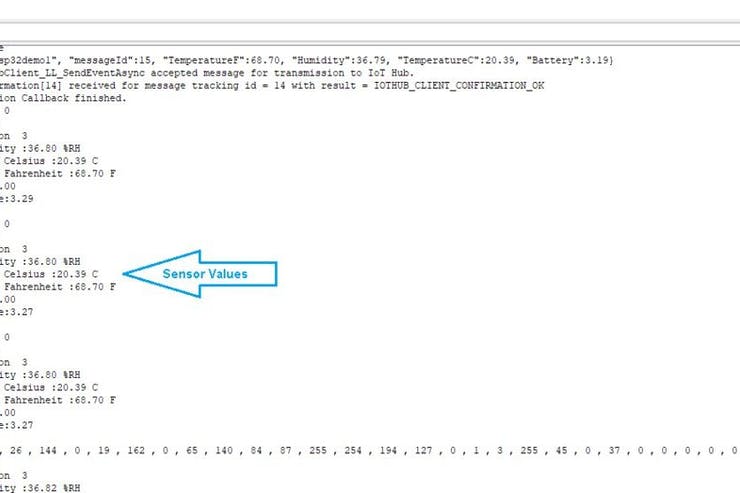
### Software Specification

* [An Azure account](https://azure.microsoft.com/en-us/)
* [Arduino IDE](https://www.arduino.cc/)

### Hardware Specification

* [ESP32](https://store.ncd.io/product/esp32-iot-wifi-ble-module-with-integrated-usb/)
* [Wireless Temperature and Vibration Sensor](https://store.ncd.io/product/industrial-long-range-wireless-temperature-humidity-sensor/)
* [Zigmo Gateway receiver](http://store.ncd.io/product/wifi-micro-gateway-microsoft-azure-iot/)

### Getting the Temperature and Humidity Values



We are getting following values from the wireless Temperature and Humidity Sensors:

* Temperature in Celcius
* Temperature in Fahrenheit
* Relative Humidity
* Battery Usage

These data is then visualized and analyzed in Azure IoT hub. To get Started with setting up of Azure IoT Hub, Go through this tutorial. The following procedure is to be followed in order to send the values Azure IoT hub.

Azure IoT hub follows MQTT protocol in order to publish and subscribe the data.

* Azure functions is another important feature provided by azure portal. Using Azure functions, we can write a piece of code or function in cloud. In this project we are parsing the JSON containing raw sensor data and getting the real temperature and Humidity values from it using Azure function. To set up Azure function follow this tutorial.
* We will get the real temperature and humidity data using parsed JSON raw data

public static async Task Run(HttpRequestMessage req, TraceWriter log)

{ double humidity;

int rawTemp;

double Ctemp;

double Ftemp;

double voltage;

string utcEnque;

string devFormat;

string utcProcess;

log.Info("C# HTTP trigger function processed a request: " + content);

JArray array = JArray.Parse($"{await req.Content.ReadAsStringAsync()}");

//parsing the JSON array

foreach(dynamic message in array){

utcProcess = message.EventProcessedUtcTime;

utcEnque = message.EventEnqueuedUtcTime;

humidity = ((message.Humid1)\*256 + (message.Humid2))/100;

rawTemp = ((message.Temp1)\*256 + (message.Temp2));

Ctemp = rawTemp /100.0;

Ftemp = Ctemp \*1.8 + 32;

int bat = ((message.Bat1)\*256 + (message.Bat2));

voltage = 0.00322 \* bat;

string utcTime = utcProcess.ToString();

DateTime localDateTime = DateTime.Parse(utcTime);

DateTime utcDateTime = localDateTime.ToUniversalTime();

string usTimeZone = "US Eastern Standard Time";

TimeZoneInfo ust = TimeZoneInfo.FindSystemTimeZoneById(usTimeZone);

DateTime dateTime = TimeZoneInfo.ConvertTime(utcDateTime, ust);

log.Info(dateTime.ToString("dd/MM/yyyy HH:mm:ss")); \

}

return req.CreateResponse(HttpStatusCode.OK, "Executed");

}

public class Message{

[JsonProperty("temp1")]

public int temp1 { get; set; }

[JsonProperty("temp2")]

public int temp2 { get; set; }

[JsonProperty("humid1")]

public int humid1 { get; set; }

[JsonProperty("humid2")]

public int humid2 { get; set; }

[JsonProperty("bat1")]

public int bat1 { get; set; }

[JsonProperty("bat2")]

public int bat2 { get; set; }

}

### Analyzing Data in PowerBi

We are using Power BI to visualize the data. It provides interactive ways to analyze data. This data can be interpreted in the form of Line charts, Bar graphs, Pie charts etc. Start by creating an account in Power Bi and sign in into your account. In the last post, we had set up Power Bi and sent data to Power Bi using stream analytics job. In this post, we are using Azure function to send the sensor data to power Bi. To set up Power Bi read this blog.

There are four methods to send the data to Power Bi:

* Directly streaming the data to Power Bi from IoT hub.
* Using API to send data to Power Bi.
* Using web-hook functionsUsing PubNub.

Here in our case we are using Power BI API and sending a HTTP response to Power BI from azure function. There are different graphs, line charts, Pi charts etc. listed in the Visualization panel. We can create the chart by selecting any of the graphs from the Visualization panel.

We can also export the data as an excel sheet or in CSV format. Which in later stages can be used for data analytics.

### Azure Function Code for PowerBI

Parse all the JSON objects from the JSON, and get the real values of temperature, humidity, and others. Here the product is a Product class object where we are storing the parsed values.

Product product = new Product();

foreach(dynamic message in array){

humidity = ((message.humid1)\*256 + (message.humid2))/100;

rawTemp = ((message.temp1)\*256 + (message.temp2));

Ctemp = rawTemp /100.0;

Ftemp = Ctemp \*1.8 + 32;

int bat = ((message.bat1)\*256 + (message.bat2));

voltage = 0.00322 \* bat;

utcProcess = message.EventProcessedUtcTime;

utcEnque = message.EventEnqueuedUtcTime;

product.Ctemperature = Ctemp;

product.Ftemperature = Ftemp;

product.humid = humidity;

product.battery = voltage;

//product.dateTime = ;

product.EventProcessedUtcTime=utcProcess;

product.EventEnqueuedUtcTime=utcEnque;

}

public class Product{

public double Ctemperature{get; set;}

public double humid{get; set;}

public double battery{get; set;}

//public double dateTime{get; set;}

public string EventProcessedUtcTime { get; set; }

public string EventEnqueuedUtcTime { get; set; }

public double Ftemperature{get; set;}}

* Now create a variable to store a connection string of Power Bi
* Create an instance of HTTP client

string connString = "https://api.powerbi.com/beta/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"; </p><p> HttpClient client = new HttpClient();

* We need to send JSON to power Bi. So, Serialize the Json using model class object.
* Send the converted JSON to power bi as an HTTP request.

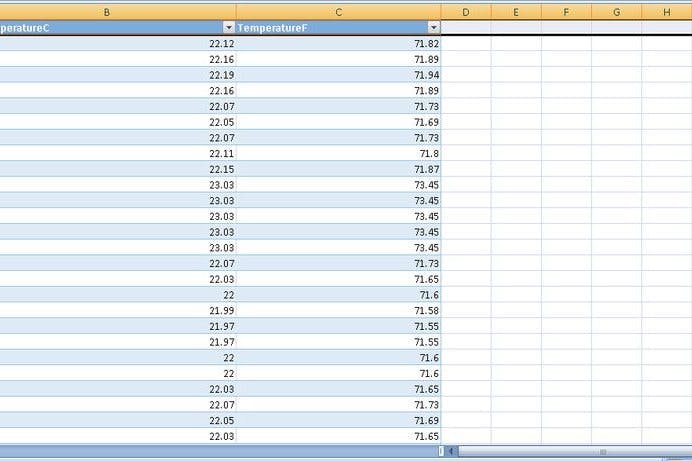
string output = JsonConvert.SerializeObject(product);

HttpContent httpContent = new StringContent("[" + output + "]");

HttpResponseMessage response = await client.PostAsync(connString, httpContent);

response.EnsureSuccessStatusCode();

### Creating Data Set

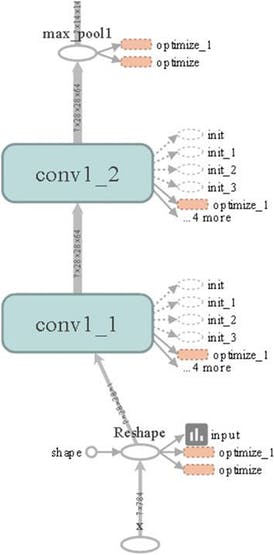


The data set consists of Temperature, Humidity values taken from the power BI in CSV format. Using this information we will predict the condition of machine. The condition of the machine is categorized in following way

* Excellent Condition
* Good Condition
* Rough Condition
* Very Rough Condition
* Danger Level
* Severe Condition

The temperature and humidity values are called as **Features** and the The condition respective to these values are called as **Labels**. We will employ feature extraction on this dataset and train this dataset to map the true labels. The Trained dataset is than compared to the test set to find the true result. This approach of machine learning is called as **Linear Regression**.

### Training the Dataset

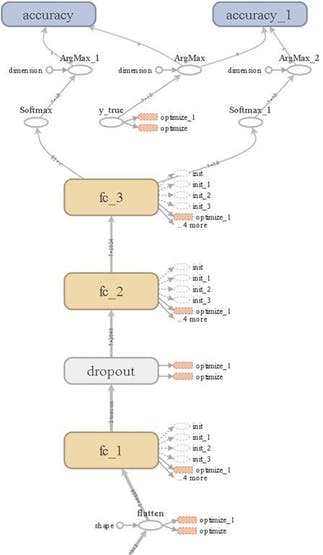


1 / 2

We have divided our data set in to two parts:

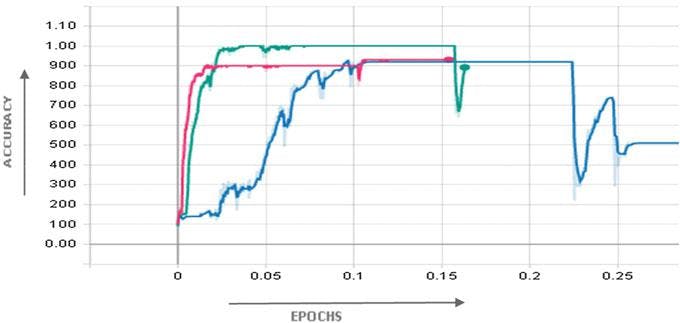
* **Training Set**- Training set contains 800 features.
* **Test Set**- The test set contains around 500 features.

We are using **LDA(Linear Discriminant Analysis)** which follows **Linear Regression** to Train the data. Once the feature are extracted using LDA. The features are provided to the Convolution layer. At this layers the weights are convoluted with the input and the biases are added to it. This process go on for different convolution layers. Then we add a max pooling layer and add a activation function(We are using **Relu** in our case more on this can be found in this research work) where the features are down sampled again to map the data with correct labels. Then at the end we have a fully connected layer where the features are again down sampled to the label size. The size of last layer is equal to the size of labels.



1 / 2

### Optimizing Accuracy and Loss



1 / 2

We optimized the data set for 2000 iteration and found the following results:

### Training Results

* **Training accuracy** (1500 epochs)- 100%
* **Training Loss**(10 epochs)- 3.4%

### Test Results

* **Test accuracy** (1500 epochs)- 98.7%
* **Test Loss** (10 epochs)- 12.5%