FINGERPRINTING BASED LOCALIZATION WITH LORA USING DEEP LEARNING TECHNIQUES

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Jait Nitinkumar Purohit

FALL
2019

FINGERPRINTING BASED LOCALIZATION WITH LORA

USING DEEP LEARNING TECHNIQUES

A Project

by

Jait Nitinkumar Purohit

Approved by:

__________________________________, Committee Chair
Xuyu Wang, Ph.D.

__________________________________, Second Reader
Xiaoyan Sun, Ph.D.

____
Date

Student: Jait Nitinkumar Purohit

I certify that this student has met the requirements for format contained in the University format manual, and this project is suitable for electronic submission to the library and credit is to be awarded for the project.

___________________________, Graduate Coordinator   ____________________
Jinsong Ouyang, Ph.D.   Date

Department of Computer Science

Abstract

of

FINGERPRINTING BASED LOCALIZATION WITH LORA USING DEEP LEARNING TECHNIQUES

by

Jait Nitinkumar Purohit

Due to an increase in Internet of Things (IoT) based applications and location-based services amongst millions of devices and gateways over the network, a lot of research is being conducted in analyzing various wireless positioning methods in Low Power Wide Area Network (LPWAN). Fingerprinting based Localization is one such technique with short-range radio frequency and addresses problems with multi-path in indoor positioning. With the increase in complex infrastructure layout and millions of devices being connected, fingerprinting approach has challenges like spatial ambiguity, long distances, low-bandwidth, scalability, cost and size constraints.

Considering these challenges and the problem at hand, this work aims at predicting accurate location using deep neural networks, for the data collected using LoRaWAN communication protocol. LoRa, a low power wide-range technology has been used to collect signal strength data at different locations from the sensors and gateways, in an indoor university building. Indoor localization using deep neural networks has achieved

1.2-2.0 [m] of mean distance error to predict accurate latitude and longitude. Additionally, to validate the comprehensive approach, a publicly available outdoor data-source collected in the city of Antwerp, Belgium is used to minimize localization error. The approach uses the neural network algorithms like Artificial Neural Network (ANN), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) in comparison with their traditional k-NN approach. The results have demonstrated that mean distance error of 191.52 [m] from LSTM has out-performed results from the k-NN algorithm. From the results, LSTM estimates sequential data more accurately, based on the user's current and previous position in the linear trajectory. Analysis of different hyper-tuning parameters of DeepML has helped to optimize the results, while reducing problems like vanishing gradient descent and overfitting in an indoor dataset.

The whole approach has been implemented in different hardware accelerators like GPU and TPU in a python-based framework called TensorFlow. Google Colab has been used to train the models. The statistical results have been plotted using Matlab. Hardware devices like Dragino LoRa gateway, Arduino, GPS LoRa shield and a DHT11 sensor have been used in this study.

_______________________, Committee Chair
Xuyu Wang, Ph.D.

____
Date

DEDICATION

To my family

Who always supported, believed and had faith in me.

To my Professor's

Who influenced and motivated me to face challenges and overcome them.

To my friends

Who kept me going in this school life and taught me never to give up.

ACKNOWLEDGEMENTS

I am thankful to Dr. Xuyu Wang, for giving me this opportunity to enhance my learnings and skills in the field of developing an IoT use-case and improving it using Machine Learning and Neural Networks. I acknowledge him for showing trust in me and allowing me to solve the challenging task related to fingerprinting based localization. I believe, he has a significant impact on my master's project because his research area in the field of IoT and Deep-Learning has also helped me to accrue more insightful knowledge aligning with my previous work-experiences in the same field. Moreover, he funded me to buy the required hardware devices for this project. He was very supportive and always gave valuable feedback during break and weekends. I would also like to appreciate Dr. Xiaoyan Sun, my second reader, for her efforts in discussing the project architecture and reviewing the report with her valuable feedback. I would also like to acknowledge Dr. Haiquan Chen for teaching me the detailed concepts of Deep Learning and AI, which has helped me build this project work. Additionally, Dr. Yuan Cheng has motivated me in the field of IoT as he was my mentor at CSU IoT Challenge, GE Predix in Spring 2018.

Lastly, this project won't have been possible without the constant support and motivation from my family, friends and faculty and staff members of the Department of Computer Science at California State University, Sacramento, for their continuous administrative support and guidance through my course curriculum.

TABLE OF CONTENTS

LIST OF TABLES

Tables Page

LIST OF FIGURES

Figures Page

CHAPTER 1

INTRODUCTION

FINGERPRINTING BASED LOCALIZATION WITH LORA USING DEEP LEARNING TECHNIQUES – is an implementation to resolve localization challenges for the Internet of Things (IoT) based applications in indoor and outdoor environments, for low-power narrowband spectrum using neural network techniques. Localization is very important in GPS based applications and hence resolving and enhancing it becomes crucial in non-linear environments. LoRaWAN is a communication protocol used in our system as it is efficient in comparison to Wi-Fi and Bluetooth technologies in such environments. This system uses neural network techniques to improve fingerprinting-based localization results instead of basic machine learning classifiers and naïve approaches discussed in various academic papers. Before digging deep into the understanding of the project let us know its importance, the existing problems, and the purpose of this project.

1.1 Importance of Fingerprinting Based Localization

Fingerprinting based Localization is common and the most efficient approach used in complex indoor and outdoor positioning applications and does not require line-of-sight measurements. With the advent of GPS based applications in the field of the edge of the Internet of Things, the positioning of edge-devices and gateways become extremely important in the context of achieving maximum computation and accurate results. In complex indoor infrastructure, fingerprinting is effective in estimating the accurate position of devices in the context of the optimal received signal strength. It is also productive in outdoor applications like car navigation, cell-phone tracking and edge-to-edge communication with cloud over multiple gateways where the object's location is necessary for services and communication. Moreover, fingerprinting based localization saves hardware installation cost and effort because of its network structure.

Fingerprinting based localization mainly consists of two main phases: offline i.e training phase and an online i.e testing phase. The offline phase involves mapping signal strength information based on LoRa gateway RSS fingerprints into the database. The online phase is the testing stage to estimate and determine the object's location based on mathematical modeling techniques like pattern matching, Euclidean distance [1], machine learning and deep learning.

1.2 Problems of Localization in Indoor and Outdoor environment

Localization in indoor and outdoor environments brings a lot of challenges due to complex infrastructure layouts, signal instability, and non-linear line-of-sight measurements. The non-linear relation between sensor nodes and target base-stations or gateways create challenges in correctly positioning. Typical and complex infrastructure layouts also lead to RSSI signal instability due to factors like height, line-of-sight, and non-line-of sight scenarios. Due to wide fluctuations of GPS and Wi-Fi signals in indoor environments lead to an inefficient localization. Spatial ambiguity [2] is a major problem where similar fingerprints are observed for distant locations in comparison to the current location. In outdoor applications where the location of the object is moving, it corresponds to a smaller number of RSSI readings, which impacts the localization accuracy [2].

1.3 LoRa based Localization techniques

LoRa stands for Long Range, is DNA for Internet of Things. LoRa is a spread spectrum modulation technique based on Chirp Spread Spectrum (CSS) technology [3]. Millions of sensor devices are interconnected using LoRa. It is effectively used for long-range communication purposes. The idea to choose LoRa over Wi-Fi and Bluetooth for fingerprinting localization is because of its efficiency and accuracy in indoor and outdoor environments. LoRa has more accurate signal strength and outperforms Bluetooth and Wi-Fi in terms of packet-drop in all line-of-sight and non-line-of-sight scenarios [3]. It also shows a higher logarithmic relation with distance in path-loss comparison with other localization techniques [3]. Therefore, LoRa is a low-power and cost-effective robust communication technology used in GPS free environments, to send small packets of secured and reliable data over long distances.

1.4 Purpose and Report Formation

The purpose of this project is to address fingerprinting based localization problems in complex indoor and outdoor environments and to design an efficient solution with LoRa technology to estimate location using deep learning techniques. This approach will provide an optimal location estimation in comparison to other mathematical approaches like pattern matching and euclidean distance. Until now we have discussed the problems which currently exist for the fingerprinting-based localization and brief overview of the LoRa technology used to overcome these problems using deep learning techniques.

The rest of the report is organized as follows. In Chapter 2, we will briefly discuss about key-concepts used in our project, like background related to various localization techniques, LoRa technology architecture vs LoRaWAN architecture, deep learning techniques like artificial neural networks, convolutional neural networks, recurrent neural network and long-short term memory and the importance of hyper-parameter tuning to optimize our deep learning results. In Chapter 3 and Chapter 4, we have discussed project requirements, and architecture followed by implementation and result from the analysis of indoor and outdoor localization in depth. The last few chapters will cover the conclusion and related work for fingerprinting based localization using other wireless techniques.

CHAPTER 2

BACKGROUND OF THE STUDY

In this section, we have introduced a brief background about various localization techniques used in many GPS-enabled applications like Range-based techniques, Range-free techniques and Fingerprinting methods for an indoor and outdoor environment. LoRa has proven efficient in our approach in comparison to BLE and Wi-Fi challenges discussed in academic research. Additionally, we will discuss how deep neural network algorithms work.

2.1 Various Localization Techniques

Localization is a well-studied topic in research areas of technology, algorithms, visualization, and evaluation. There are a lot of technologies that are accurate and ubiquitous to achieve localization. Localization techniques are of three types,

- Range-based algorithms,
- Range-free algorithms,
- Fingerprinting.

Range-based techniques rely on distance between end-nodes to predict the target location [4]. The approaches used in range-based algorithms are proximity, lateration, and hyperbolic-lateration and angulation. Distance is estimated using the time of flight (ToA) and received signal strength (RSS) [4]. Range-free algorithms focus on the target object’s proximity to transmit beacons with known locations [4]. Range-free localization requires simple hardware in comparison to the former counterpart. Fingerprinting is a mapping solution to address localization problems with multipath scenarios [4]. It has a short-range RF propagation pattern and has an offline (testing) and an online (training) phase. Fingerprinting requires a dense site survey and is usually better for symbolic localization [4]. It is more efficient and accurate in estimating localization, using received signal strength measurements.

In our system, we have used a fingerprinting based localization technique with LoRa technology, discussed in brief in the following sub-sections.

2.2 LoRaWAN Architecture

The fourth industrial revolution is heavily dependent on Internet-of-Things networks where billions of devices are interconnected over the internet. The challenges related to these devices are efficient power, cost-constraints and the ability to communicate reliably over longer ranges. Using LPWAN, one can address these issues in their IoT applications.

LoRaWAN is a widely used proprietary for long-range communication based on chirp spread spectrum modulation technique called LoRa developed by LoRa Alliance [1]. LoRa operates in different frequency bands in different geolocations viz. Europe region has a frequency band of 863 to 870 MHz, US region has 902 to 928 MHz while China operates between 779 to 787 MHz Symbols are being encoded using the number of chirps, which causes the signal to be spread over various bandwidth of channels. This technique helps to reduce interference with other signals. Spreading Factor (SF) determines the number of chirps i.e. time of packet arrival and can range from 7 to 12. The value of SF closer to 12 means longer ranges can be achieved at the expense of low data rate in comparison to low spreading factors. The relation between SF and signal range has a direct effect on the received signal strength and distance mapping [1], [5].

As shown in Figure 1, sensor-devices communicate with single or multiple gateways or base-stations over a single hop, provides a TCP/IP uplink (sensor to gateway) to LoRaWAN cloud server like The Things Network, in our project. The network server is mainly responsible for passing messages between edge devices and applications [5]. Any type of communication in this network architecture is encrypted twice, i.e. once using a network session key and the other using application session key [5]. Microservices to handle gateway traffic can be written using publisher and subscriber model using Apache ActiveMQ – MQTT broker.

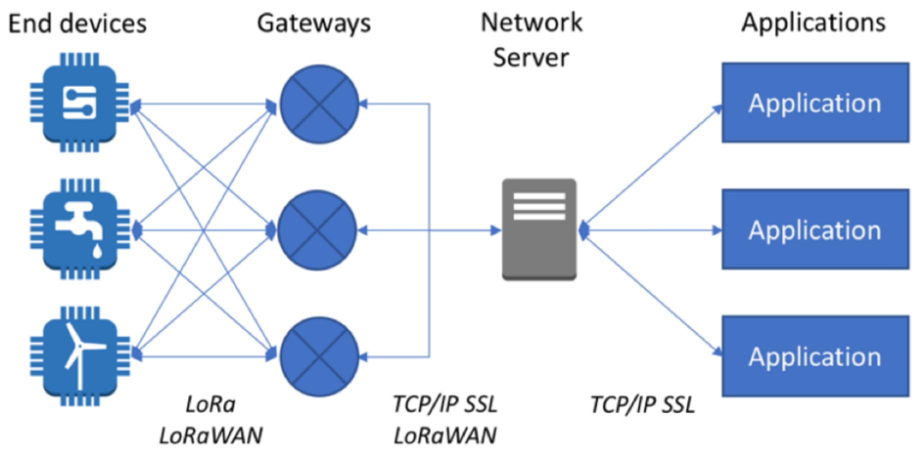


Figure 1: Sensor-Gateway interaction over LPWAN technology

Image credits: Craig Lawton

Compared to other communication technologies like Sigfox and NB-IoT, LoRa works with higher bandwidth which helps in enabling localization with Time Difference of Arrival (TDoA) [1]. Therefore, accurate synchronization amongst receiving gateways is very important. End-nodes are physical hardware sensor devices that contain sensing capabilities and computing power up to some extent. Gateways, also known as base-stations or an access-points are used to pick up all message payloads from edge devices. These payloads or RF packets are converted to IP packets (array of bits) over the Network server and are further sent as traditional IP networks to the Application server. The application server is the place where the actual IoT application is residing and is useful with data collected from edge devices [5]. Application servers can run mostly on the cloud to perform advanced analytics, data preprocessing and create RESTful web-services.

LoRaWAN is a secured and reliable communication protocol, as it encrypts data packets using 128-bit Network_Session_Key at the network server level and further encrypts using 128-bit App_Session_Key at application level [6]. Internally, it uses AES algorithms for security authentication. LoRa operates in the sub-GHz frequency spectrum and so it prevents other signals to penetrate and addresses multi-path issues which are important in the localization problem, that we are addressing in this study.

2.3 Artificial Neural Networks

Artificial Neural Networks are a class of machine learning where the computation of neuron is taken place internally and the networks are used to have inter-connectivity amongst them [7]. Neurons from the current layer receive input from previous layers for further computation along with weights adjusted. The connection between these interconnected neurons is identified by weights and learning parameters, which are used to update the parameters of the network to get a favorable output. This type of networks has feed-forward neural networks where the first layer is an input layer, followed by hidden layers and an output layer. Data is transferred using hidden layers from input to the output layer. There are several parameters like activation functions, optimizers, epoch and batch-size used to train artificial neural network models.

2.4 Deep Neural Networks

Deep Neural Networks are also a class of machine learning algorithms which relies on non-linear processing neurons for feature extraction. [7] Deep Neural networks requires specific type of hardware accelerators like GPU or TPU, RAM, physical memory and storage depending on the complexity of the problem at hand. They have multiple hidden layers that help to model complicated functions. Non-linear data scenarios are usually handled by DNN and are useful in big-data use-cases. DNN is used in the field of artificial intelligence for computer vision, natural language processing, audio and video processing [7], [8].

2.5 Recurrent Neural Networks

Recurrent Neural Networks or RNNs are a type of ANNs where output data is dependent on input data as well as historical data. These data have a sequential relation. Moreover, they have networks with loops which helps to persist the information. RNN has a chain-like structure which exchanges information to successors [2]. The main idea behind recurrent neural networks is to allow us to operate over sequences of input, output or both types of data in a recurrent fashion. The hidden layer is the most important layer here as it remembers the information about the sequence of data. RNN can be used in localization approaches as the current node data will be related to previously collected data in the trajectory [2]. RNN, in general, are used for speech recognition and natural language processing. There are few disadvantages of using RNN, which are discussed below,

- Vanishing gradient descent problem
- Training long sequences of data is a challenge using ReLu and Tanh as an activation function

To address these issues of RNN, a novel algorithm LSTM was invented, which has been discussed in the following sub-section.

2.6 Long-Short Term Memory

Long short-term memory is used to deal with sequence prediction problems. They have an edge in comparison to conventional feed-forward neural networks and RNN in many ways, as they can remember patterns for longer durations of time. LSTMs avoids long-term dependency problems, resulting into better accuracy than RNN.

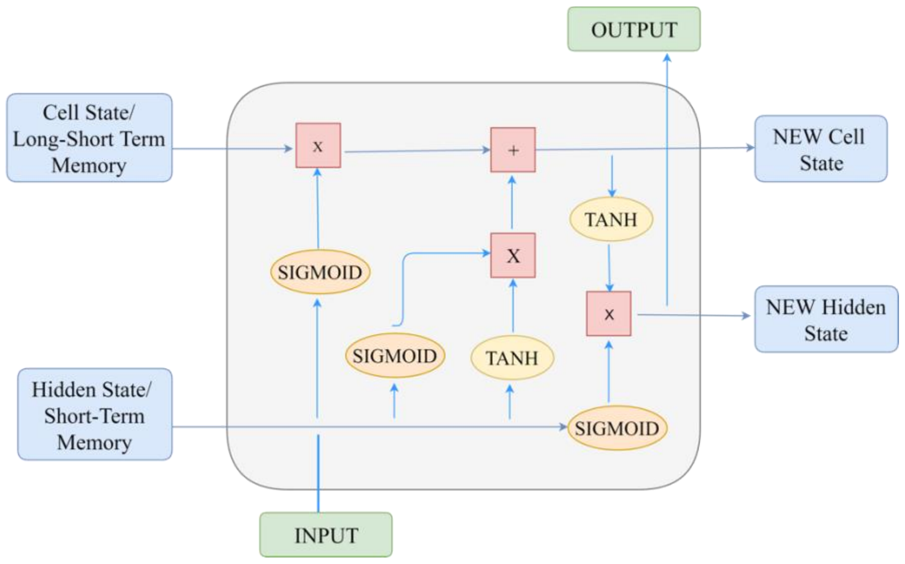


Figure 2: Long-Short Term Memory Architecture with 4 interacting layers

Image credits: Gabriel Loye

LSTMs also have a chain-like architecture and have a slightly different structure for the repeating module as described in Figure 2. The cell state and the horizontal line on the top of each cell are the most important part of LSTM. [9] The cell state is similar to a conveyor belt and has limited linear interactions through the whole chain. There is a gate-like structure that helps to add or remove information in the cell state. These gates allow the information to flow through and are composed of a sigmoid as an activation function to perform pointwise multiplication operation [9]. The sigmoid function ranges between zero and one and entails about how much information each element should be allowed to flow in [9].

In our implementation of LSTM, to optimize fingerprinting localization, we have created an x-array and y-array matrix. These matrices are returned as numpy arrays while calling sequence () function. These sequence functions are set to size equals to 1, which helps in giving results for latitude and longitude for each edge device. LSTM model is sequential in nature and has a linear stack of layers. This model can be passed with input_shape argument to the first layer as well as the type of layer i.e. dense.

```
model.add(LSTM(64,dropout=0.1, recurrent_dropout=0.1, input_shape=(1, 6)))
```

Dropout is a regularization method where recurrent connections to LSTM and input are removed from activation and weight changes while training the network. Dropout is used here to avoid the reducing overfitting and improving model performance in case of indoor localization as we have very fewer data-points. The main advantage of using LSTM over RNNs is the problem of vanishing gradient descent being re-solved using memory gates. Though there are issues related to overfitting, as dropout technique is hard to implement. Long-short term memory algorithm is used mostly in the field of natural language processing due to its characteristics.

2.7 Convolutional Neural Network

As shown in Figure 4, Convolutional Neural Networks or CNN is a classification algorithm and also a class of deep-learning algorithms which are usually used to deal with image recognition, digit recognition or object recognition. It takes an image as an input and assigns weights and biases to the image by breaking it into parts and learn accordingly to predict an accurate output image. The main architecture of CNN is similar to that of neurons in a human brain [10]. Figure 3 shows an input matrix or image are passed as an input to the convolutional kernel, which then performs matrix multiplication to extract high-level features from the input image [10].

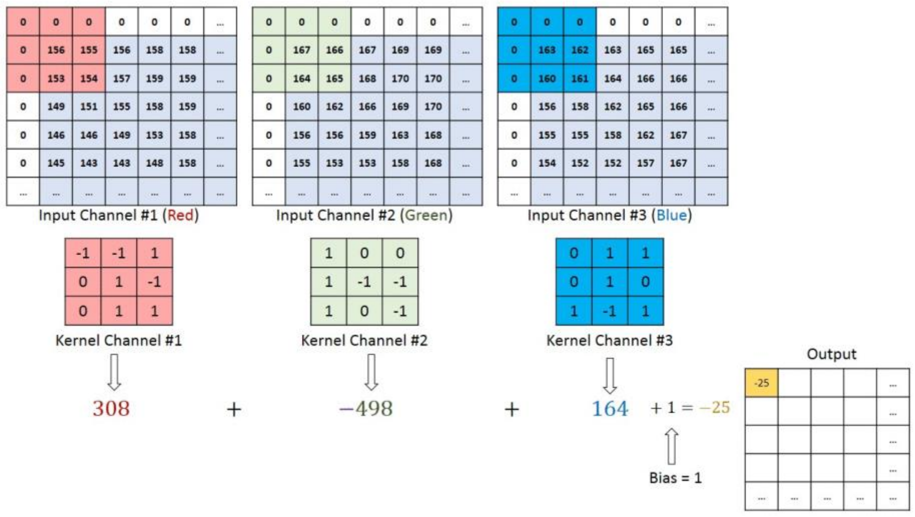


Figure 3: Convolution Operation on an MxNx3 image matrix with a 3x3x3 Kernel

Image credits: Sumit Saha

The other concepts in CNN are related to padding, which helps to preserve the dimensions of input in the output label. The Pooling layer is similar to the Convolutional Layer which is helpful to reduce the spatial dimensions of the network by creating feature maps. It provides an approach to down-sampling [10]. Max-pooling and average pooling are different types of pooling layers in CNN architecture.

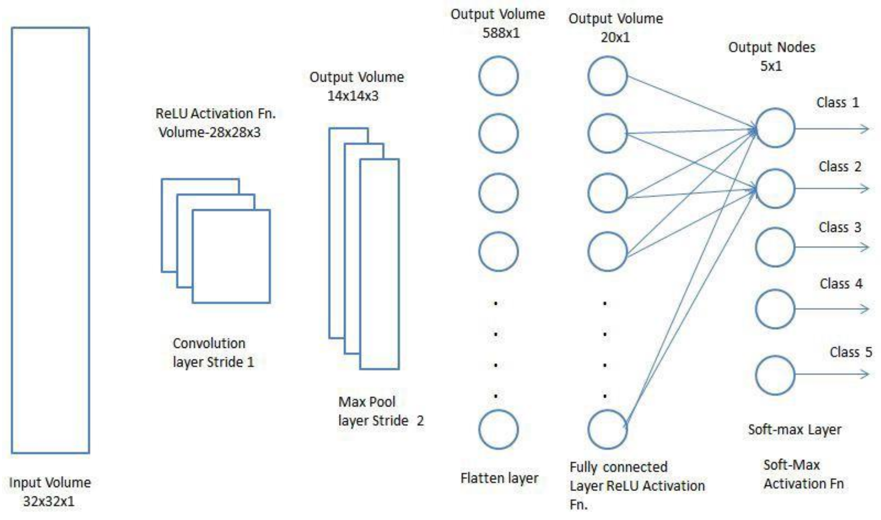


Figure 4: Convolutional Neural Network Architecture
Image credits: Sumit Saha

In our implementation, we have used CNN as a regression problem to estimate the mean location error in meters.

2.8 Importance of Hyperparameter Tuning

Hyperparameter tuning [11] is a part of model optimization to minimize the testing error. Choosing the correct number and diversity of these parameters is dependent on each classifier and can vary accordingly. We have implemented hyperparameter tuning to neural network models to order to minimize localization error by using different permutations and combinations of optimal parameters like batch-size, learning-rate, optimizers, activation functions, and hidden layers.

CHAPTER 3

PROJECT REQUIREMENTS

The indoor fingerprinting-based localization requires a specific setup in terms of hardware and software being pre-configured in the cloud, The Things Network. This approach is divided into two phases: an offline (training) phase and an online (training) phase [26]. In the offline phase, RSSI values are collected for each predefined location on the 3rd Floor of RVR, a university building at California State University. The data-points are stored in a database called ***fingerprints***. The online phase is a testing phase where RSSI values are known and testing locations points like latitude and longitude are unknown. Using deep-learning techniques, one can estimate these testing locations and the mean location error. The training of the model is done using [12] TensorFlow framework from Google, Keras [13] and Scikit-Learn [14] libraries. Additionally, Google Colab has been used as a free cloud service to train these models as it provides hardware accelerators like GPU and TPU [15]. The hardware required for this experiment is LoRa Dragino Kit [16] which has Arduino UNO [16], LoRa Gateway, different sensors, LoRa GPS shield and required cables for connections. The software part of LoRa setup have been done in C, while model training and evaluation has been done in python [17]. We have discussed Indoor experimental setup and additional project requirements in the following sections.

3.1 Indoor Experimental Setup

Indoor Localization was implemented using a specific setup using Dragino LoRa gateways and a sensor node sending data payloads at different training and testing locations. We have chosen RVR, 3rd Floor, to carry out our experiments. The data-collection strategy has been carried out, dividing the floor into virtual (X, Y) coordinate system, so as to collect 2D data in all the directions of the lobby as a part of offline phase, as shown in Figure 5, Dragino LoRa LG01 gateways, also known as base-stations, have been configured and set up in the RVR 3rd floor lab, at a static location. The black arrow in the below Figure 5, indicates the user's walking trajectory while sending the DHT11 sensor data to these gateways at different locations. The horizontal lobby was 28 [m] long, while vertical lobby 8.5 [m] long. The training and testing data are randomly collected within 1-3 meters of difference.

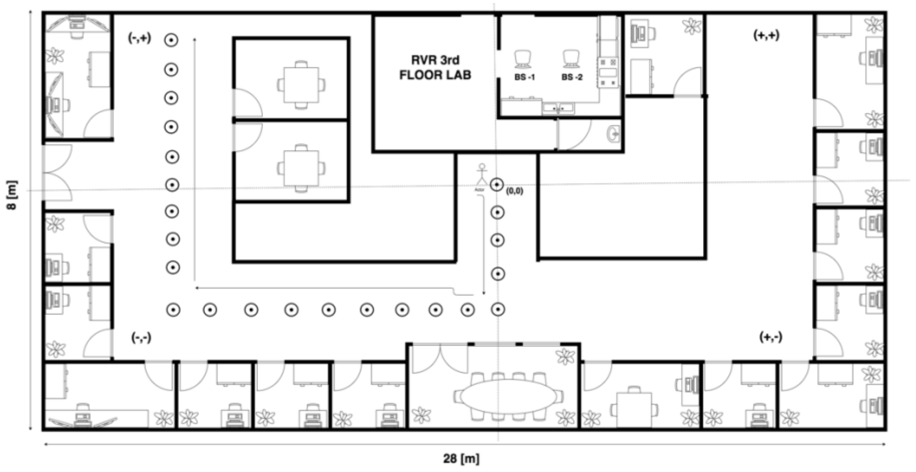


Figure 5: Floor Map of Indoor Localization test site

The RSSI (Received Signal Strength Indicator) value is the most important indicator or measurement of the signal strength of a received radio signal. The values are measured in dBm and can have values between 0 dBm (excellent strength) to -120 dBm (extremely poor). The value of RSSI is dependent on many factors like,

- Antenna type of the gateway that is transmitting
- Number of obstacles like walls in an indoor environment
- Non-linear distance between gateway and devices

The other features that we are measuring in the process are spreading factor (SF) which is the duration of the packets received and horizontal dilution of precision (HDOP) measuring the GPS signal quality using satellite configuration. The lower value of HDOP is better in terms of its geometric quality. Latitude and Longitude are noted in (X, Y) coordinate format.

3.2 Dragino LoRa Gateway Setup with Arduino UNO

LoRaWAN, a long-range communication protocol has been used in our indoor fingerprinting-based localization approach due to its efficiency in comparison to Bluetooth and Wi-Fi in line-of-sight and non-line-of-sight scenarios. It is more secure and reliable than other wireless technologies as it can transmit data in encrypted format over longer distances.

The hardware required for our setup has been listed below.

- 2 x LG01-N Dragino Single Channel LoRa Gateways
- 1 x LoRa GPS/Shield & Arduino UNO
- 1 x DHT11 Temperature and Humidity Sensor
- DuPont Wires (male to male, female to female, female to male)

The experimental setup steps are listed in chronological order.

3.2.1 Setup LoRa Node with Arduino to send uplink-downlink message to LoRa gateway

Connect LoRa node (DHT11 Sensor) with Arduino using jumper wires as shown in Figure 6. Also, connect Dragino LoRa gateway with the antenna facing upwards and keep it in a fixed position.

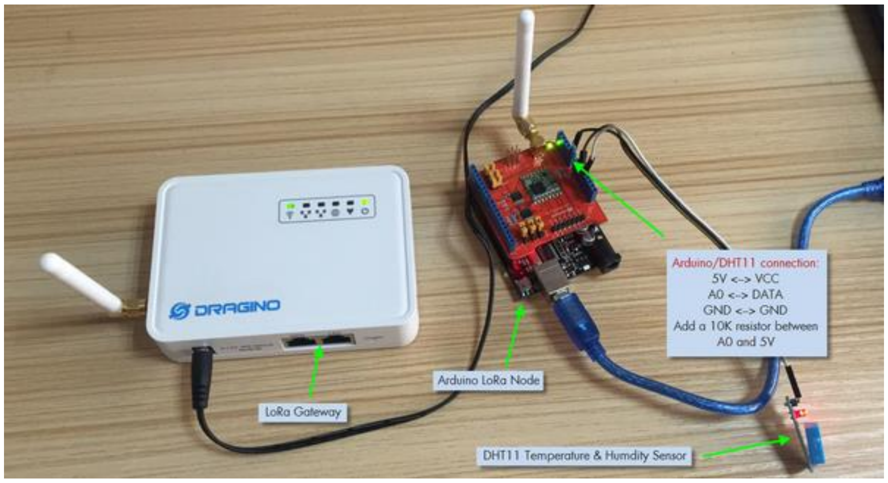
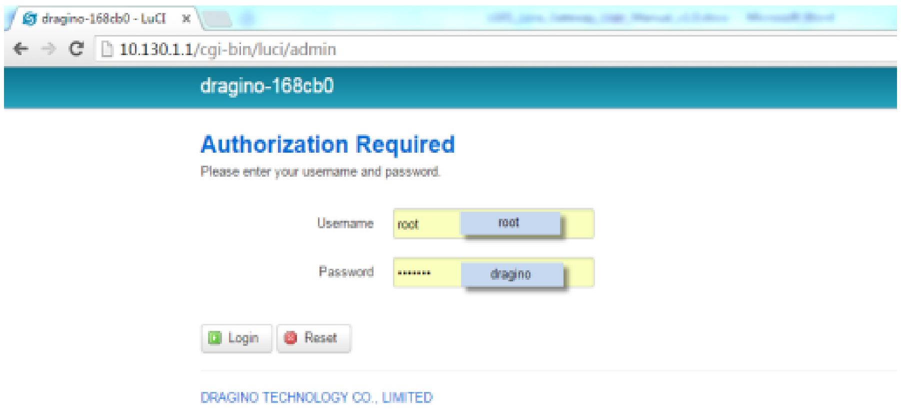

Arduino/DHT11 connection:
5V <-> VCC
A0 <-> DATA
GND <-> GND
Add a 10K resistor between A0 and 5V
Arduino LoRa Node
LoRa Gateway
DHT11 Temperature & Humdity Sensor
DRAGINO

Figure 6: Connect LoRa Gateway with LoRa Node via Arduino

3.2.2 LoRa Gateway Authorization Screen

Open browser and type http://10.130.1.1/cgi-bin/luci/admin. This will open the Authorization page for this gateway. Enter the username as ‘root’ and password as ‘dragino’ as credentials as shown in Figure 7.



Authorization Required

Please enter your username and password.

Username: root

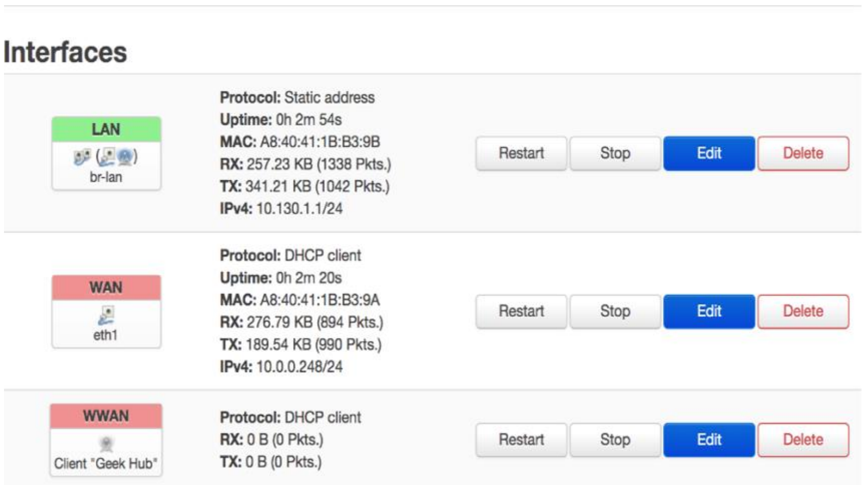
Password: ••••••• dragino

DRAGINO TECHNOLOGY CO., LIMITED

Figure 7: LoRa Gateway Authorization Screen

3.2.3 LoRa Gateway Connectivity – (WAN)

Once you authorize using user credentials, go to the network page and select appropriate wireless AP. One can also join via WAN connection. Additional configurations are required to set up like WPA paraphrase, Name of the network and assign firewall-zone i.e. WAN in our case. Make sure to save and apply your changes after configuring. One can check the interfaces section after setting it up, as shown in Figure 8.



LAN br-lan
Protocol: Static address
Uptime: 0h 2m 54s
MAC: A8:40:41:1B:B3:9B
RX: 257.23 KB (1338 Pkts.)
TX: 341.21 KB (1042 Pkts.)
IPv4: 10.130.1.1/24

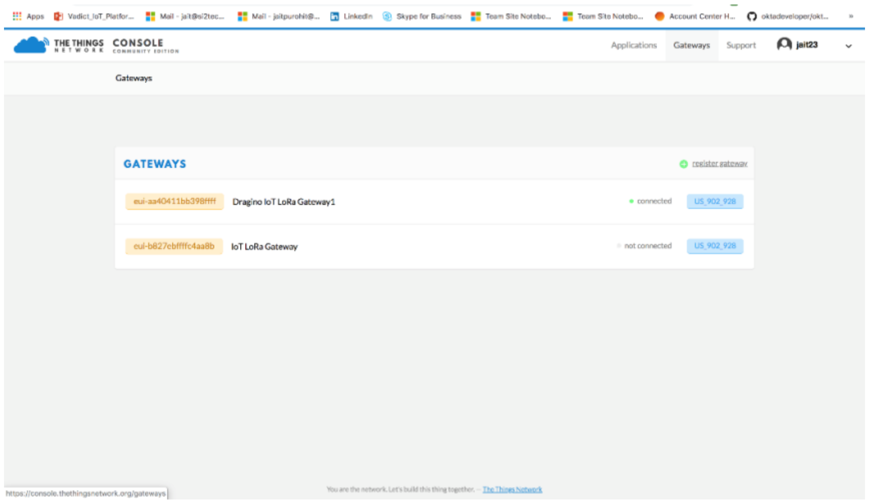
WAN eth1
Protocol: DHCP client
Uptime: 0h 2m 20s
MAC: A8:40:41:1B:B3:9A
RX: 276.79 KB (894 Pkts.)
TX: 189.54 KB (990 Pkts.)
IPv4: 10.0.0.248/24

WWAN Client "Geek Hub"
Protocol: DHCP client
RX: 0 B (0 Pkts.)
TX: 0 B (0 Pkts.)

Figure 8: Setup Connectivity on LoRa Gateway

3.2.4 Register Gateway on The Things Network Cloud (TTN)

Open browser and type https://account.thethingsnetwork.org/users/login. This will open the Authorization page for The Things Network, a cloud service provider for gateway configuration and monitoring. Register and login using appropriate credentials. Every LG01-N gateway has a unique id that can be found below. Create and set up a new gateway under Gateways section with details like gateway-id, frequency i.e. 915 MHz is USA region, along-with latitude, and longitude of the lab location. The status will be not connected right now.



eui-aa40411bb398ffff	Dragino IoT LoRa Gateway1	connected	US_902_928
eui-b827ebffffc4aa8b	IoT LoRa Gateway	not connected	US_902_928

Figure 9: Register Gateway on The Things Network Cloud (TTN)

Now, our next step is to go ahead and configure server settings on the dragino admin portal.

- IoT Service: LoRaWan/RAW forwarder
- Debug Level: Little message output
- Service Provider: The Things Network
- Server Address: ttn-router-us-west

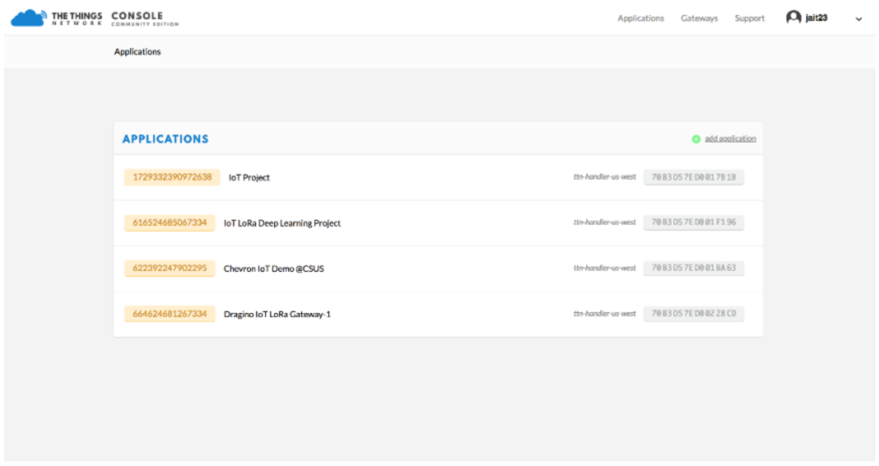
- Server Port: 1700
- Latitude and Longitude: 38.42 and -121.24

Configure Radio Frequency Settings:

- Frequency (Unit/Hz): 9150000000 (USA Region)
- Spreading Factor: SF7
- Coding Rate: 4/5
- Signal Bandwidth: 125 kHz

Now check the status of the gateway being connected.

3.2.5 Register Device on The Things Network



APPLICATIONS

1729332390972638	IoT Project	ttn-handler-us-west	70 B3 D5 7E D0 01 7B 18
616524685067334	IoT LoRa Deep Learning Project	ttn-handler-us-west	70 B3 D5 7E D0 01 F1 96
622392247902295	Chevron IoT Demo @CSUS	ttn-handler-us-west	70 B3 D5 7E D0 01 BA 63
664624681267334	Dragino IoT LoRa Gateway-1	ttn-handler-us-west	70 B3 D5 7E D0 02 28 CD

Figure 10: Register Application on The Things Network Cloud (TTN)

This step registers application details on The Things Network as shown in Figure 10. This is a meta-data configuration step required for setting up devices and gateways on the cloud. Every application will have a unique identifier called app_eui. One can register multiple applications on TTN. It also represents the region in which your application is hosted, i.e. ‘ttn-handler-us-west’.

3.2.6 Register Device on The Things Network

Our next step is to connect the DHT11 sensor i.e. Humidity and Temperature sensor to the LoRa shield as a hat on top of Arduino Uno as shown in Figure 11. We are powering the Arduino UNO using a blue USB cable. After setting this, we will set up the LoRa shield and UNO as an OTTA sensor device in TTN.

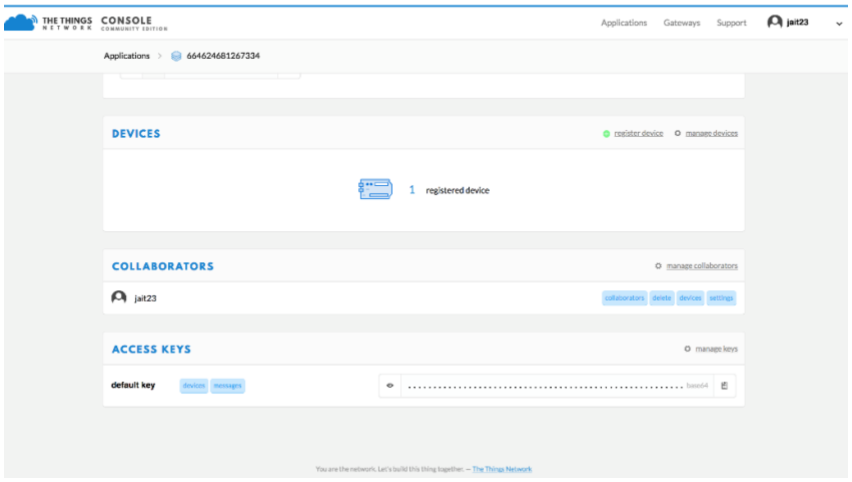
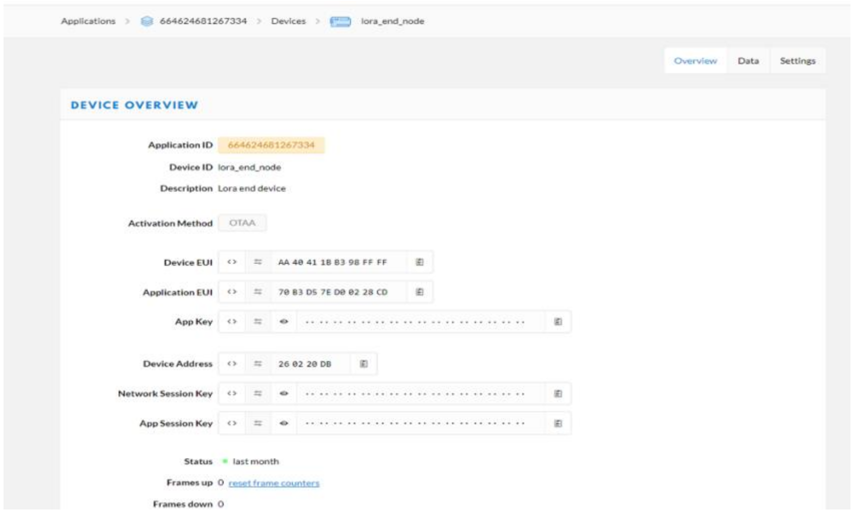


Figure 11: Register LoRa end node on The Things Network Cloud (TTN)

3.2.7 Setup LoRa Device Node Configurations

After registering the device node, we will set up the LoRa shield and UNO as an OTTA sensor device in TTN. As shown in Figure 11, we are configuring device properties. There are two types of activation methods viz. OTAA (Over-The-Air Activation) and ABP (Activation by Personalization). To achieve basic security mechanism in LoRa we have used OTAA, while ABP has been used for prototyping use-cases. OTAA also needs Device EUI i.e. gateway unique id, Application EUI and App Key in LSB format that needs to be configured into the Arduino Sketch as our next step.



Applications > 664624681267334 > Devices > lora_end_node

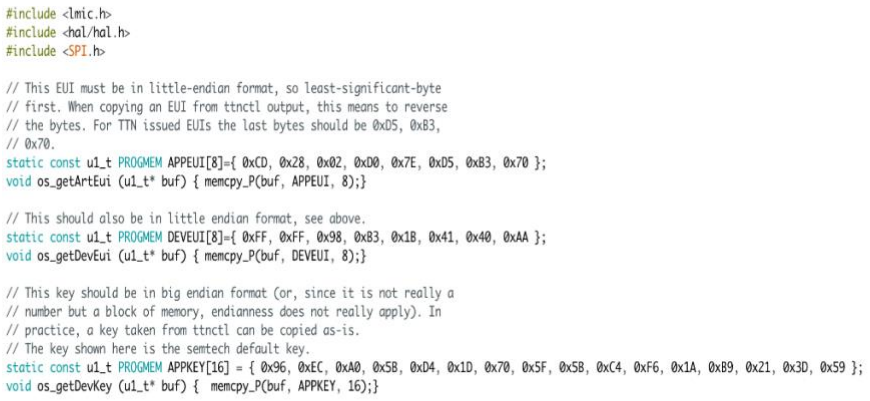
Overview | Data | Settings

DEVICE OVERVIEW

Application ID	664624681267334
Device ID	lora_end_node
Description	Lora end device
Activation Method	OTAA
Device EUI	AA 40 41 1B B3 98 FF FF
Application EUI	70 B3 D5 7E D0 02 28 CD
App Key	•• •• •• •• •• •• •• •• •• •• •• •• •• •• •• ••
Device Address	26 02 20 DB
Network Session Key	•• •• •• •• •• •• •• •• •• •• •• •• •• •• •• ••
App Session Key	•• •• •• •• •• •• •• •• •• •• •• •• •• •• •• ••
Status	● last month
Frames up	0 reset frame counters
Frames down	0

Figure 12: Setup LoRa Device Node Configurations

3.2.8 Code Snippet to configure device node in Arduino



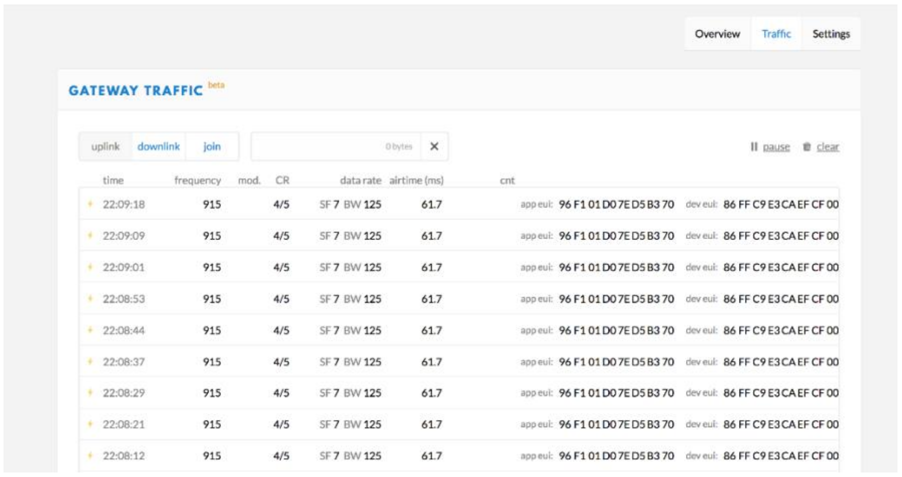
```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8]={ 0xCD, 0x28, 0x02, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]={ 0xFF, 0xFF, 0x98, 0xB3, 0x1B, 0x41, 0x40, 0xAA };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
static const u1_t PROGMEM APPKEY[16] = { 0x96, 0xEC, 0xA0, 0x5B, 0xD4, 0x1D, 0x70, 0x5F, 0x5B, 0xC4, 0xF6, 0x1A, 0xB9, 0x21, 0x3D, 0x59 };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
```

Figure 13: Code Snippet to setup input keys in Arduino Sketch – LoRa Node

The source code can be found at https://github.com/jait-23/deep-learning-lora-rssi/blob/master/Arduino/arduino-lmic-master/examples/ttn-otaa/ttn-otaa.ino

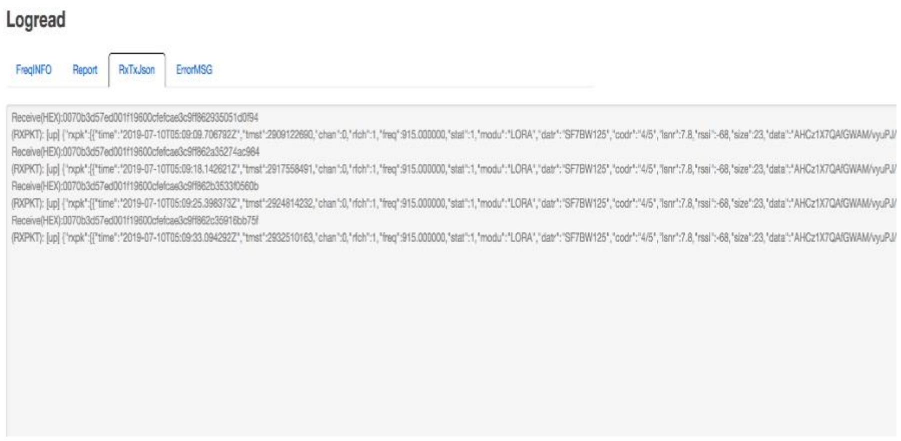
3.2.9 Collect Signal Strength Data from Server Logs

One can monitor gateway traffic on the Things Network as shown in Figure 14, with important information like data-rate, SF, bandwidth, frequency, and payload information. The received signal strength information related to the gateway can be found under Log-read on its server, as shown in Figure 15.



time	frequency	mod.	CR	data rate	airtime (ms)	cnt		
22:09:18	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:09:09	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:09:01	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:08:53	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:08:44	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:08:37	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:08:29	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:08:21	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00
22:08:12	915		4/5	SF 7 BW 125	61.7		app eui: 96 F1 01 D0 7E D5 B3 70	dev eui: 86 FF C9 E3 CA EF CF 00

Figure 14: Uplink-Downlink gateway traffic data retrieval



```
Receive(HEX):0070b3d57ed001f19600cfefcae3c9ff862935051d0f94
(RXPKT): [up] {"rxpk":[{"time":"2019-07-10T05:09:09.706792Z","tmst":2909122690,"chan":0,"rfch":1,"freq":915.000000,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","lsnr":7.8,"rssi":-68,"size":23,"data":"AHCz1X7QAfGWAM/vyuPJ/
Receive(HEX):0070b3d57ed001f19600cfefcae3c9ff862a35274ac984
(RXPKT): [up] {"rxpk":[{"time":"2019-07-10T05:09:18.142621Z","tmst":2917558491,"chan":0,"rfch":1,"freq":915.000000,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","lsnr":7.8,"rssi":-68,"size":23,"data":"AHCz1X7QAfGWAM/vyuPJ/
Receive(HEX):0070b3d57ed001f19600cfefcae3c9ff862b3533f0560b
(RXPKT): [up] {"rxpk":[{"time":"2019-07-10T05:09:25.398373Z","tmst":2924814232,"chan":0,"rfch":1,"freq":915.000000,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","lsnr":7.8,"rssi":-68,"size":23,"data":"AHCz1X7QAfGWAM/vyuPJ/
Receive(HEX):0070b3d57ed001f19600cfefcae3c9ff862c35916bb75f
(RXPKT): [up] {"rxpk":[{"time":"2019-07-10T05:09:33.094292Z","tmst":2932510163,"chan":0,"rfch":1,"freq":915.000000,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","lsnr":7.8,"rssi":-68,"size":23,"data":"AHCz1X7QAfGWAM/vyuPJ/
```

Figure 15: JSON Response of Signal Strength Data from Gateway server logs

CHAPTER 4

PROJECT ARCHITECTURE AND TECHNOLOGIES USED

4.1 System Architecture

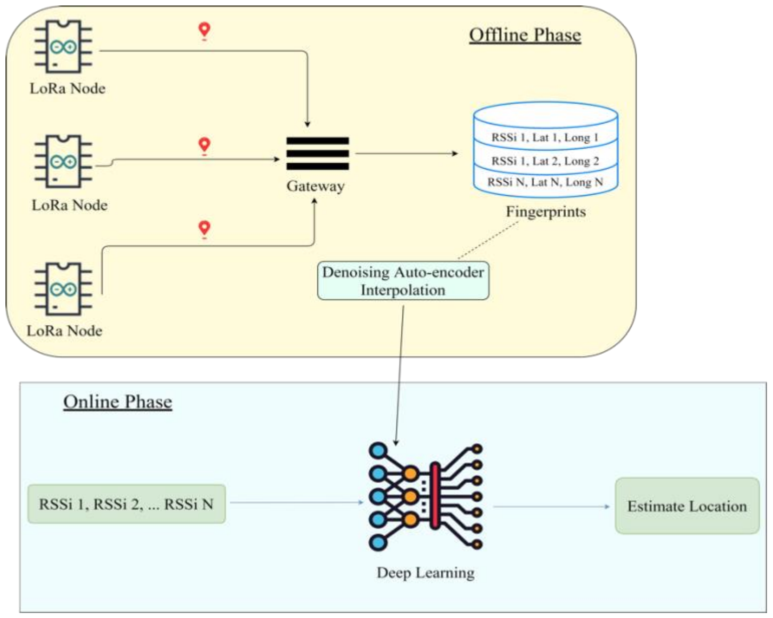


Figure 16: System Architecture of LoRa based Localization using DeepML Techniques

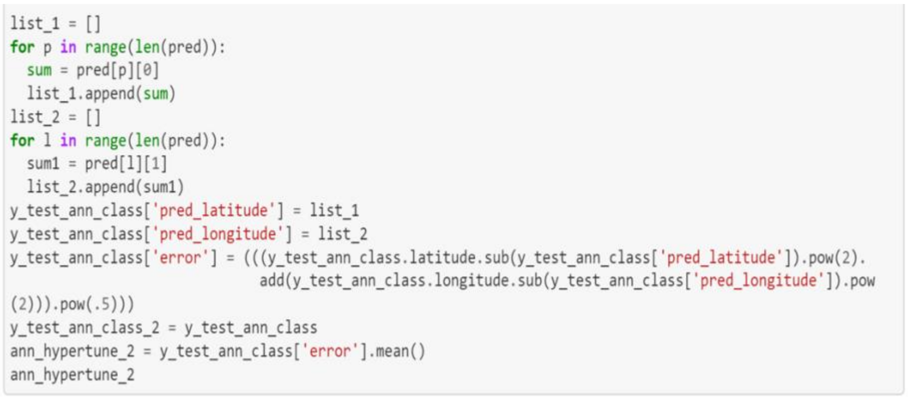
The underlying architecture of the LoRa based localization using DeepML techniques is shown in Figure 16, focusing on the high-level architectural components. A LoRa sensor node sends data payloads to multiple Gateways, with the user's pre-defined latitude and longitude position as a data payload. The Received Signal Strength is an important indicator to measure the signal quality of base-stations or gateways at the receiver end. The other information like spreading factor (SF) and horizontal dilution of precision (HDOP) are also collected at the Gateway endpoint and stored in a time-series database as a part of training (offline) phase. These data-points are also called as ***fingerprints***.

At each location, 5 instantaneous RSSI readings were collected as a part of the offline phase. The data collected at the location for multiple times have observed different RSSI values, on a number of occasions. The reasons for instability and fluctuations of RSSI values in indoor environments are listed below,

- Dynamically changing environments or human obstacles
- Interference from other devices being connected to Wi-Fi
- Receiver-Antenna Orientation
- Experiments carried out during peak hours at University
- A Non-linear relation between sensor node and gateway as distance changes

Due to the problems discussed above, a novel approach of LoRa technology and Neural Networks helps to address non-linear multipath issues. Deep Learning algorithms like LSTM and RNN handle gaussian noise better than basic machine learning algorithms. LSTM handles sequential data efficiently for fingerprinting based localization as it compares the user's current location with the user's previous location in the same trajectory. Moreover, as the size of sensor data increases, it becomes difficult to train models using basic machine learning models. Deep Learning models handle issues related to spatial ambiguity and RSSI instability. Denoising Auto-encoders have been leveraged as a part of the training phase to extract real information of the data outliers. This interpolation technique of imputation is a data preprocessing and data filtering method to corrupt the data on purpose and changing those values to zero. Auto-encoders is a type of artificial neural networks, used to extract encoded data information in an unsupervised manner and decode true features of the dataset. The training and test data have been split into a 70%-30% ratio for model training. The online phase or testing phase is all about predicting location information using Deep Learning Models, like artificial neural networks, convolutional neural networks, and long-short term memory [24]. Our goal is to minimize the mean location error using these algorithms and optimizing our localization results using hyperparameter tuning by reducing the testing error.

The Loss function used here is Mean Squared Error, which calculates the mean of the square of differences between two latitudes and two longitudes points i.e. lat1, lat2, long1, long2. This has been implemented using the mathematical concept of Euclidean distance formula in pandas as shown in Figure 17. The below code snippet also iterates through the predicted and test, latitude and longitude values. It appends the sum to list_1 and list_2 which will be used to calculate mean squared error using Euclidean distance formula.



```
list_1 = []
for p in range(len(pred)):
  sum = pred[p][0]
  list_1.append(sum)
list_2 = []
for l in range(len(pred)):
  sum1 = pred[l][1]
  list_2.append(sum1)
y_test_ann_class['pred_latitude'] = list_1
y_test_ann_class['pred_longitude'] = list_2
y_test_ann_class['error'] = (((y_test_ann_class.latitude.sub(y_test_ann_class['pred_latitude']).pow(2).
                              add(y_test_ann_class.longitude.sub(y_test_ann_class['pred_longitude']).pow
(2))).pow(.5)))
y_test_ann_class_2 = y_test_ann_class
ann_hypertune_2 = y_test_ann_class['error'].mean()
ann_hypertune_2
```

Figure 17: Code Snippet to calculate Mean Location Error using Euclidean Distance Formula

4.2 Technology Used

For this project, I have used two coding languages, multiple frameworks, libraries and tools to implement fingerprinting-based localization with LoRa using deep learning.

The languages that have been used here are Python 3.7 for training and evaluating models and C for Arduino Uno. The hardware required for this project is all included in a single-channel Dragino LoRa kit available online [16].

Software Stack:

Given below are the list of technologies and languages used for this project along with the purpose of them.

Table 1: Software Technologies Used

Name of Technology	**Purpose of its Usage**
Numpy	Data Preprocessing
TensorFlow [12]	Machine Learning Framework
Keras [13]	Neural Network API
Scikit-Learn [14]	Machine Learning Library
Google Colab [15]	Python Cloud Platform – Training Deep Learning Models on GPU & TPU as hardware accelerators
The Things Network [16]	Open-source, Decentralized cloud network
Python 3.7 [17]	Programming Language – Python libraries and ML code
C [18]	Programming Language – Arduino Sketch Development
Matlab [19]	Plot Results using CDF functions, graphs etc.
Pandas [20]	Data Preprocessing
Matplotlib [21]	Data Visualization

CHAPTER 5

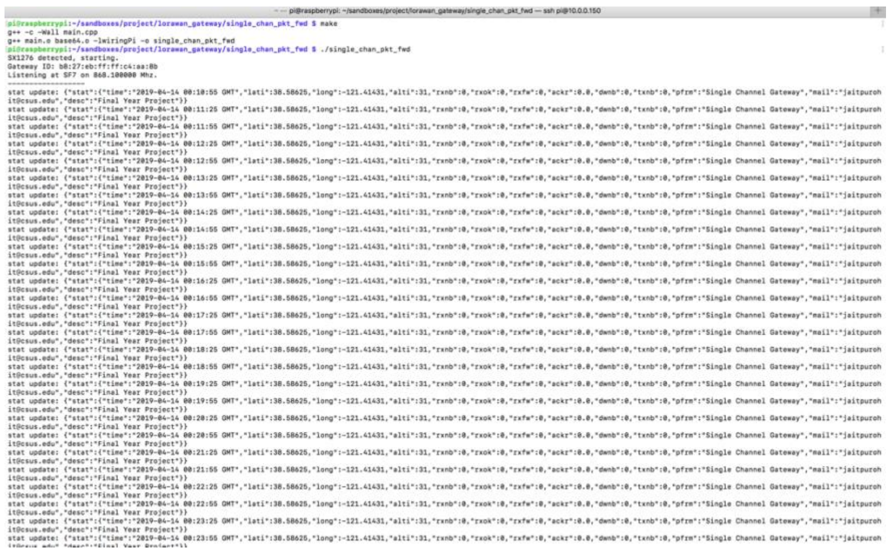
PROJECT IMPLEMENTATION AND RESULTS

The chapter highlights critical aspects of the implementation of LoRa based Localization using Deep Learning techniques, showing and explaining screenshots of the final results along with a useful section of code snippets. All the functionality is covered in detail, emphasizing the interaction between the offline phase or training phase where data is collected and preprocessed and the online phase where model training, evaluation to estimate location error has been implemented. The Deep Learning techniques discussed are supervised learning based on continuous multi-label prediction of latitude and longitude.

5.1 Data Collection

The first and foremost step in Machine Learning is the Data on which implementation will be carried. Understanding the domain of the data is very important before implementing the solution.

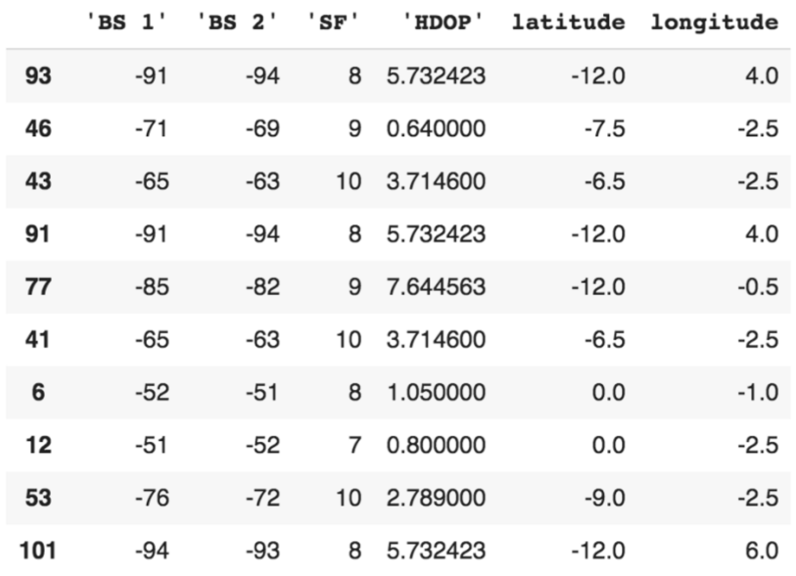
The indoor localization approach has 2 LoRaWAN base-stations and 1 DHT sensor sending data payloads via Arduino UNO and LoRa Shield to those gateways. The signal strength also known as RSSI has been collected from these gateways in this phase. 2D data has been collected in (X, Y) format from Riverside University building, 3rd Floor, across the horizontal and vertical lobbies. Gateways are kept static in the lab whereas sensor node sends data along-with user's walking trajectory. The packet RSSI data from Single Channel LoRa gateway can be logged into the database from the logs as shown in Figure 18. The other features were spreading factor (SF) ranging from 7-12 and horizontal dilution of precision (HDOP). The data collected imported into CSV format and will be a part of data import for our next phase of machine learning, i.e. data preprocessing which will help us understand its domain.



```
pi@raspberrypi: ~/sandboxes/project/lorawan_gateway/single_chan_pkt_fwd — ssh pi@10.0.0.150
pi@raspberrypi:~/sandboxes/project/lorawan_gateway/single_chan_pkt_fwd $ make
g++ -c -Wall main.cpp
g++ main.o base64.o -lwiringPi -o single_chan_pkt_fwd
pi@raspberrypi:~/sandboxes/project/lorawan_gateway/single_chan_pkt_fwd $ ./single_chan_pkt_fwd
SX1276 detected, starting.
Gateway ID: b8:27:eb:ff:ff:c4:aa:8b
Listening at SF7 on 868.100000 Mhz.
------------------
stat update: {"stat":{"time":"2019-04-14 00:10:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:11:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:11:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:12:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:12:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:13:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:13:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:14:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:14:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:15:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:15:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:16:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:16:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:17:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:17:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:18:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:18:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:19:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:19:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:20:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:20:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:21:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:21:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:22:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:22:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:23:25 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
stat update: {"stat":{"time":"2019-04-14 00:23:55 GMT","lati":38.58625,"long":-121.41431,"alti":31,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"jaitpuroh
it@csus.edu","desc":"Final Year Project"}}
```

Figure 18: Indoor Data collected from Single Channel LoRa Gateway logs

The Localization process have two vertical lobbies and one horizontal lobby where this experiment was carried. The two base-stations were kept at a fixed position in the lab and configured over the server. The offline data for every location was recorded into CSV. The user’s current and previous location had a difference of 1-3 [m] of range. As 2D data was collected, the latitude was kept fixed and longitude was a variable for horizontal lobby. On the contrary, longitude was kept fixed and latitude was a variable for vertical lobby. For each location, 5 reading were taken and a total of 250 messages were stored in offline data and imported as data-frame using pandas, as show in Figure 19. This was our indoor localization data collection stage.



	'BS 1'	'BS 2'	'SF'	'HDOP'	latitude	longitude
93	-91	-94	8	5.732423	-12.0	4.0
46	-71	-69	9	0.640000	-7.5	-2.5
43	-65	-63	10	3.714600	-6.5	-2.5
91	-91	-94	8	5.732423	-12.0	4.0
77	-85	-82	9	7.644563	-12.0	-0.5
41	-65	-63	10	3.714600	-6.5	-2.5
6	-52	-51	8	1.050000	0.0	-1.0
12	-51	-52	7	0.800000	0.0	-2.5
53	-76	-72	10	2.789000	-9.0	-2.5
101	-94	-93	8	5.732423	-12.0	6.0

Figure 19: Data-frame for Indoor Localization data

On the contrary, the outdoor approach is from publicly available LoRaWAN dataset [1]. The data has been collected over a period of 3-4 months from Antwerp, a city in Belgium. The goal of their approach was to create a benchmark to evaluate localization in outdoor environments using the kNN approach. 123,529 LoRaWAN messages from several nodes represent our data frame rows. 68 base-stations have been used as a gateway to transfer data from devices to the application layer using LoRa network layer. Figure 20 shows how outdoor data is spatially scattered over a larger radius in the city area.

They used twenty cars for this purpose carrying the Firefly X1 GPS receiver. This receiver continuously mustered latitude and longitude information of the car. The other information like HDOP and SF were also recorded. This information was sent as a LoRaWAN message using IM880B module [1].

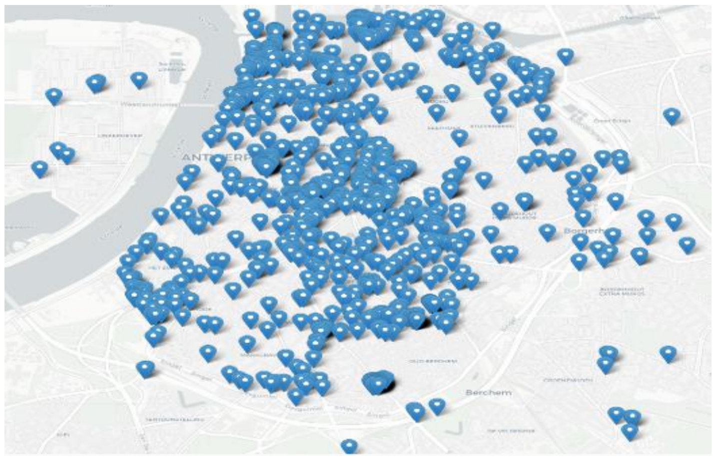


Figure 20: Spatially scattered LoRaWAN messages from Outdoor Data

These messages were sent frequently because of wider bandwidth and higher data rate using LoRa. They have used a callback function to forward the data payload to data server with another network information attached. 68 base-station RSSI values are stored in database with their timestamp. The other columns are related to spreading factor, HDOP, and geo-coordinates.

5.2 Exploratory Data Analysis

Exploratory Data Analysis is an important field of Data Science and Data Analytics when it comes to big data and understanding the correlation amongst each feature. These data are generated from various applications like images, enterprise, retail, autonomous cars and billions and trillions of interconnected devices from IoT applications. To understand these data in 2-D, 3-D or N-D, it becomes difficult with such a big chunk of data. Dimensionality Reduction methods like Principal Component Analysis (PCA) helps to get more relevant information [21]. Moreover, the perturbation rank matrix is used to analyze weights from each feature on the output label. Matplotlib and Seaborn are the two python libraries that we have used in this project. The only installation requirement was to execute a pip install command and then importing them into the source code. Figure 21 shows the usage of matplotlib where 2D correlation heatmap is plotted. It takes values from the first dimension as rows of the data-frame and columns are represented using the second dimension. It helps to understand incident patterns and detect anomalies and outliers.

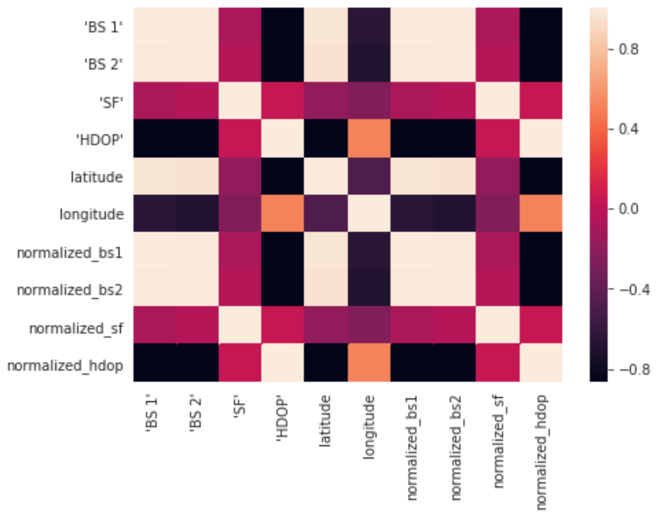


Figure 21: 2D Correlation Matrix Heatmap

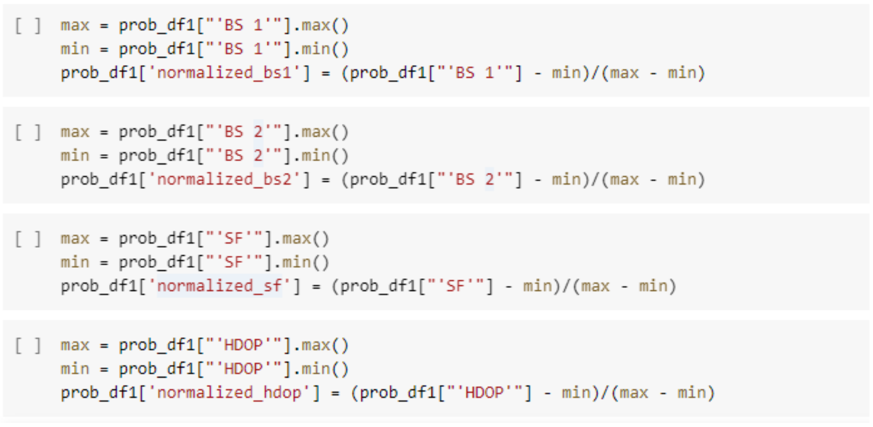
5.3 Data Preprocessing

Data preprocessing is one of the most important pre-requisites to train any kind of machine learning or deep learning model. It requires checking and evaluating some basic conditions which are described as follows:

- Import necessary libraries – Pandas and Numpy,
- Import CSV dataset,
- Handle missing or duplicate data, if any,

- Handle outliers, by plotting statistical Normal Distribution Curve,
- Normalize the data columns – Min-Max function as shown in Figure 22,
- Shuffle Data – to avoid any bias in model training,
- Splitting the data into Training and Testing set – 70% train and 30% test.

Additionally, there are noisy data and outliers in our use-case due to multi-lateration issues in an indoor and outdoor environment. We have leveraged a neural network technique, denoising auto-encoders [22], to handle these data as a part of data-pre-processing. This has been explained in the following sections along-with other common interpolation techniques.



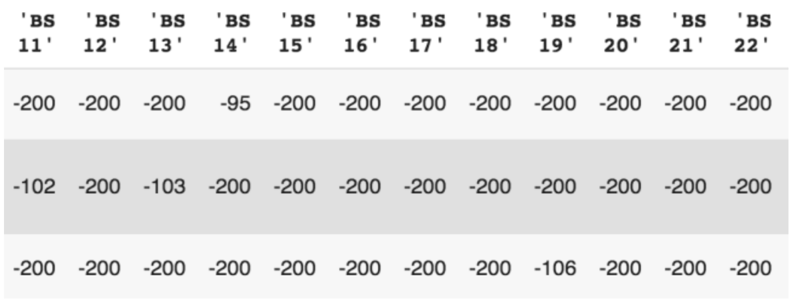
```
[ ]  max = prob_df1["'BS 1'"].max()
     min = prob_df1["'BS 1'"].min()
     prob_df1['normalized_bs1'] = (prob_df1["'BS 1'"] - min)/(max - min)

[ ]  max = prob_df1["'BS 2'"].max()
     min = prob_df1["'BS 2'"].min()
     prob_df1['normalized_bs2'] = (prob_df1["'BS 2'"] - min)/(max - min)

[ ]  max = prob_df1["'SF'"].max()
     min = prob_df1["'SF'"].min()
     prob_df1['normalized_sf'] = (prob_df1["'SF'"] - min)/(max - min)

[ ]  max = prob_df1["'HDOP'"].max()
     min = prob_df1["'HDOP'"].min()
     prob_df1['normalized_hdop'] = (prob_df1["'HDOP'"] - min)/(max - min)
```

Figure 22: Normalizing using the min-max function

5.4 Common Interpolation Techniques

Lora signals from edge-devices cannot be measured for all the locations. Therefore, the outlier values have to be interpolated using sample data points. Several interpolation techniques like linear, cubic, quadratic and denoising auto-encoders are discussed below.

As we have known the fingerprinting technique can be divided into two phases, offline and online [26]. We have also seen that the message payloads have information like RSSI, ID, timestamp, spreading factor (SF), HDOP and geolocation co-ordinates for each edge device. Each base-station in the multi-lateration process have different signal strength values and are sent to the online phase and are stored with their corresponding coordinates. There are many outliers in the process and can be interpolated using different approaches.



'BS 11'	'BS 12'	'BS 13'	'BS 14'	'BS 15'	'BS 16'	'BS 17'	'BS 18'	'BS 19'	'BS 20'	'BS 21'	'BS 22'
-200	-200	-200	-95	-200	-200	-200	-200	-200	-200	-200	-200
-102	-200	-103	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-200	-200	-200	-106	-200	-200	-200

Figure 23: Outlier Values before Interpolation

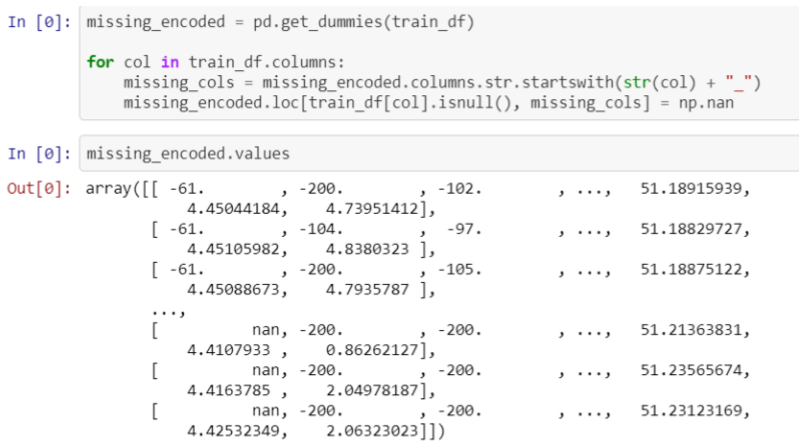
Linear Interpolation used here is a method of fitting the curve using linear polynomials to form new data points within some discrete range of known data points. Linear interpolation for (x_0, y_0), (x_1, y_1) ... (x_n, y_n) are defined as the concatenation of interpolants in linear order for each pair of data points [23]. Linear Interpolation in the machine learning world is training with missing values by reconstructing the outliers using the curve of the input/output [23]. Similarly, we implemented cubic and quadratic interpolation which have a thin line of difference compared to linear interpolation. Quadratic interpolation is called degree two polynomials, and cubic is degree three polynomials. Therefore, base-stations with missing values are interpolated using forward or backward pattern of known data. We compared the interpolated results of LoRa signals with linear, cubic and quadratic interpolation functions in python SciPy inbuilt library functions.

We have compared our interpolation results with denoising autoencoders in the next section. They are a stochastic version of basic autoencoders in deep learning.

5.5 Interpolation Using Denoising Autoencoders

Autoencoders are a part of deep learning neural networks which are mainly used for selection and extraction. Moreover, there are many nodes in the hidden layer than there are inputs, which are called as “Identity Function”, also called a “Null Function”. Denoising autoencoders [22] solves our problem by corrupting the outlier data on purpose by converting them to null values. Theoretically, the percentage of input nodes that are being set to null values is about 50%, which depends on the amount of data and input nodes.

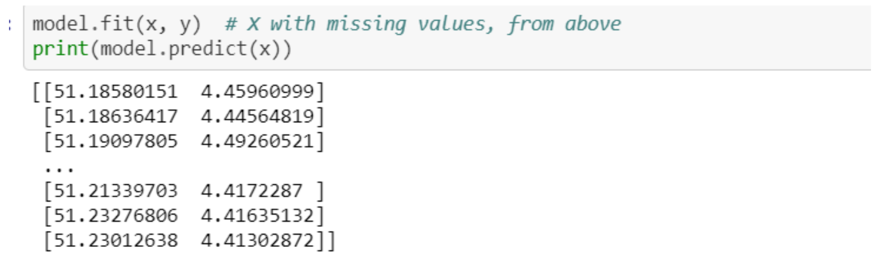
In our implementation of denoising autoencoders, we have a function to shuffle data around and learn more about the data by attempting to reconstruct it. This process of shuffling has helped to recognize the features within the noise and will allow us to classify the input values. While training the neural network, it generates a model and measures the distance between the benchmark that has been set and the model through a loss function f(n). The loss function f(n) attempts to reduce the loss function by resampling the shuffled inputs and re-constructing the data-value until it finds those inputs which are true to the actual value.



```
In [0]: missing_encoded = pd.get_dummies(train_df)

        for col in train_df.columns:
            missing_cols = missing_encoded.columns.str.startswith(str(col) + "_")
            missing_encoded.loc[train_df[col].isnull(), missing_cols] = np.nan

In [0]: missing_encoded.values

Out[0]: array([[ -61.        , -200.        , -102.        , ...,   51.18915939,
                  4.45044184,    4.73951412],
               [ -61.        , -104.        ,  -97.        , ...,   51.18829727,
                  4.45105982,    4.8380323 ],
               [ -61.        , -200.        , -105.        , ...,   51.18875122,
                  4.45088673,    4.7935787 ],
               ...,
               [         nan, -200.        , -200.        , ...,   51.21363831,
                  4.4107933 ,    0.86262127],
               [         nan, -200.        , -200.        , ...,   51.23565674,
                  4.4163785 ,    2.04978187],
               [         nan, -200.        , -200.        , ...,   51.23123169,
                  4.42532349,    2.06323023]])
```

Figure 24: Code snippet for one-hot encoding on corrupted values

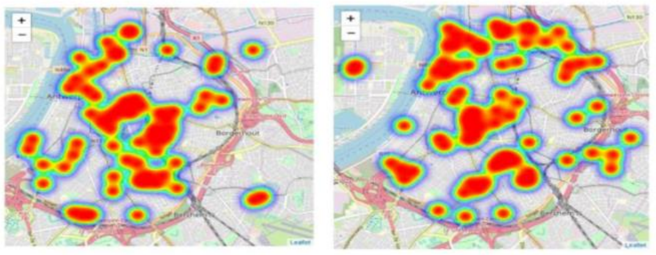
Figure 24 shows the code snippet for one-hot encoding on corrupted values using pandas get_dummies(). Figure 25 shows the code snippet to predict those corrupted values using denoising autoencoder model.

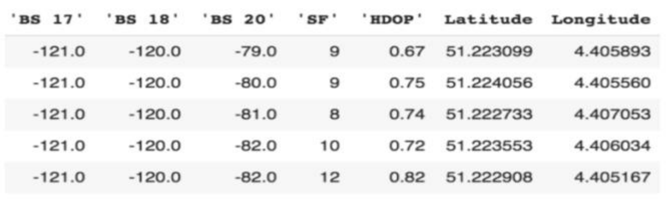


```
: model.fit(x, y)  # X with missing values, from above
  print(model.predict(x))

[[51.18580151  4.45960999]
 [51.18636417  4.44564819]
 [51.19097805  4.49260521]
 ...
 [51.21339703  4.4172287 ]
 [51.23276806  4.41635132]
 [51.23012638  4.41302872]]
```

Figure 25: Code Snippet to predict noisy data

Figure 26 shows the results from multiple base-stations after interpolation using denoising autoencoders along-with the heatmap. Therefore, the whole process attempts to find the risk of identity-function by converting outliers to null values and autoencoders must then denoise or learn to reconstruct it back, minimizing the log-loss function.





'BS 17'	'BS 18'	'BS 20'	'SF'	'HDOP'	Latitude	Longitude
-121.0	-120.0	-79.0	9	0.67	51.223099	4.405893
-121.0	-120.0	-80.0	9	0.75	51.224056	4.405560
-121.0	-120.0	-81.0	8	0.74	51.222733	4.407053
-121.0	-120.0	-82.0	10	0.72	51.223553	4.406034
-121.0	-120.0	-82.0	12	0.82	51.222908	4.405167

Figure 26: Interpolation results using Denoising Auto-encoders

Table 2 shows the comparison of interpolation results of f(x,y) using various interpolation techniques from 100 random points for each of the two base-stations. The original data is divided into 70% training and 30% testing datasets. After interpolation, fingerprinting maps for each base-stations have been generated for training data. The difference between the original RSSI value and interpolated RSSI value has been calculated for each test data. Denoising Auto-encoders have given better results in terms of lower average and standard deviation from its interpolated data in comparison to other common interpolation techniques. We also compared our results with that from fingerprinting maps and denoising auto-encoders performed well in extrapolated areas.

Table 2: Comparison of Interpolation methods

Gateway	**Interpolation Algorithm**	**Average**	**Standard Deviation**
BS-1	Linear	7.85	7.52
	Cubic	10.12	9.23
	Quadratic	12.47	10.63
	Denoising Autoencoder	6.52	3.79
BS-2	Linear	6.97	7.17
	Cubic	9.24	8.47
	Quadratic	10.47	9.37
	Denoising Autoencoder	5.62	3.47

5.6 Academia's Outdoor Localization Approach using kNN

This approach describes the basic fingerprinting kNN algorithm discussed in [1]. Firstly, the dataset is divided into 70% training, 15% test and 15% evaluation. Secondly, The Euclidean distance matrix is created for training and evaluation set. The *k*-nearest neighbors are identified for each evaluation. For each value of *k*, the authors are calculating the mean location estimate, finding the optimal value of *k.* A similar process is executed for training and test set. Finally, the author validates the best value for *k,* by evaluating all the errors.

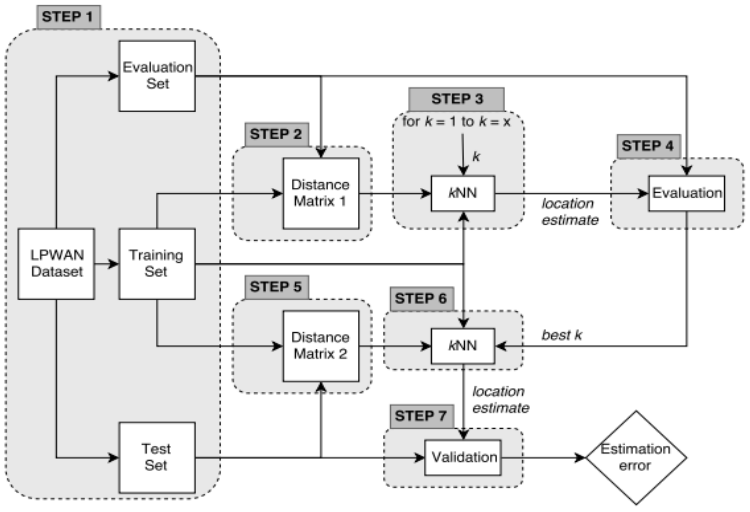


Figure 27: Fingerprinting algorithm using k-NN [1]

The mean location error using this algorithm was 398.4 [m] and is considered very poor due to computational complexity cost for very large data size. Therefore, using this approach leads to more cost and signal congestion problems in a crowded spatial outdoor environment. Using neural networks for such non-linear, sequential and noisy data reduces mean location error using fingerprinting based localization with LoRa technology.

5.7 Improvements using Neural Networks for Outdoor Localization

We removed outliers and noisy data using the interpolation technique, denoising auto-encoders as discussed in previous sub-sections. Normalization has helped to compare data columns within one specific range and has an unbiased impact on the output column set i.e. latitude and longitude. Neural network models have performed better in terms of accuracy and log-loss scores than basic ML techniques. Trying different permutation and combinations of activation functions, optimizers and tweaking the neuron counts for hidden layers, changing batch-size have produced better accuracy. The running time of these classifiers is more in comparison to basic machine learning algorithms as it takes more time to run training data segregated into batches and multiple hidden layers with more neuron counts.

Evaluating the results from deep-learning models, it is clearly observed from Figure 28, that deep-learning models out-performs basic machine learning models like kNN results from [1], and LSTM as our best model giving mean location error of 191.5276 [m] with 64 neuron counts, ReLu as activation function and Adam as an optimizer with a batch-size set to 128, epochs as 10 and dropout as 0.1 to avoid overfitting problems. Our loss function is the mean squared error. For each LSTM model, sequence_size was kept to 1. In many cases, the training of the model stopped after the 5th iteration as the model stopped performing well. The validation split was 0.3, and the batch size was tweaked to remove biases and variance in the model. The training loss was 0.1360 and validation loss was 0.0911 when the model stopped its training after 5th iteration.

Artificial neural networks (ANN) and the Convolutional neural network (CNN) were other neural network models used, which performed better than the basic machine learning model. The best ANN model achieved 284.78366 [m] with a batch size of 256 and epoch size of 20. On the other hand, CNN with batch size 81 and epoch set to 3, achieved 215.06 [m] mean location error. Additionally, the mean and standard deviation of LSTM model is 191.52 [m] and 112.74 [m].

Table 3: Outdoor Localization: Mean Location Error [m] results using DeepML

Deep ML Method	Nodes	Epoch	Batch Size	Dropout	Mean Location Error [m]
ANN	100	10	64	None	284.78475
	30	20	256	None	284.78366
	90	10	512	None	284.81696
LSTM	64	10	512	0.1	191.52726
	128	12	256	0.5	194.77348
CNN	512	3	64	0.3	215.06072
	512	3	81	0.5	221.75332

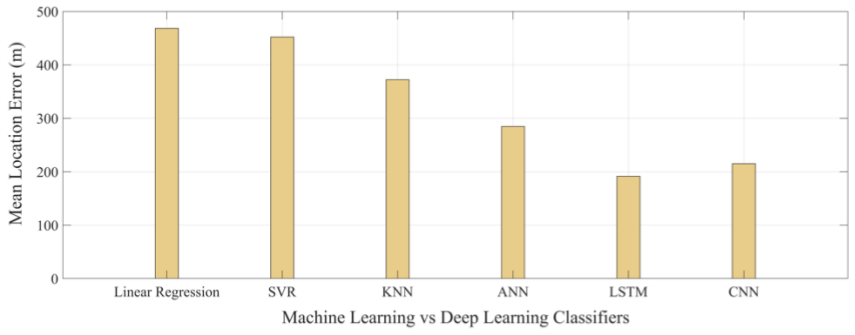


Figure 28: Outdoor Localization results using DeepML

5.8 Impact of design parameters on LSTM for Outdoor Localization

Due to hyperparameter tuning, we were able to optimize the model accuracy and reduce the testing error. The model parameters like batch-size, type of optimizer, activation functions, test-data size, dropout layer and hidden layers are very important in tuning our model performance. Choosing the right value of these parameters depends on many factors discussed below,

- Number of hidden-layer neurons

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

Figure 29: Formula for counting the number of hidden units

Ni = Input Neurons, **No** = Output Neurons, **Ns** = Training Samples, **α** = Scaling Factor (2-10)

- Dropout layer – to prevent from overfitting
- Activation/Transfer function – Sigmoid, ReLu, Tanh, SoftMax
- Optimizers – Adam, RmsProp, SGD
- Batch-Size – 64, 81, 128, 256, 512, 1024
- GPU vs TPU training accuracy

The results we have gotten so far ran on GPU hardware accelerator. We have compared its results with the deep-learning training done on google-colab's TPU accelerator. Our experiments were running on a smaller convolutional neural network and we saw our results were skewed as a custom network is not optimized for TPUs. Our dataset had approximately 120K records, divided into 90K training samples and 30K testing samples. We ran our experiments as described herein as different scenarios. One such scenario has LSTM modeled with 3 hidden layers, dropout set to 0.1, with 10 epochs and activation function being sigmoid. Another scenario has CNN with 3 hidden layers, 2 dense layers, with 3 epochs. The conclusion was, running the model on TPU achieved better results in the case of a larger dataset.

We evaluated the proposed LSTM models with test data size of 100, 200, 300, 400 and 500, to verify how varying test data size impacts the mean location error of outdoor localization on LSTM [24]. The epoch, hidden units, optimizer, activation function, and batch size, dropout layer are set to 20, 3, Adam, ReLu, 64 and 0.1 respectively. As shown in Figure 30, distance errors of both scenarios are almost the same for different test data sizes. Specifically, the mean distance errors for regular LSTM and hyper-tuned LSTM are 191.52 [m] and 194.77 [m], respectively, for test data size of 200. Adding more data samples and training locations could help us achieve higher precision [24].

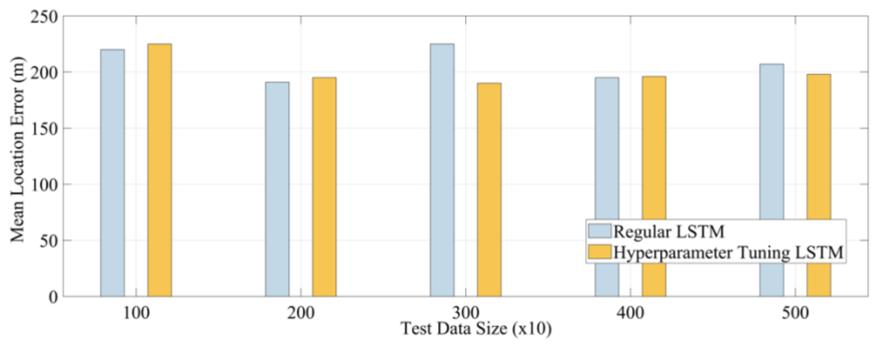


Figure 30: Impact of test data on mean location error

Batch size plays an important role in optimizing the model’s accuracy. Smaller batch-sizes has limited parallelization of stochastic gradient descent in deep learning [24]. Larger batch sizes are used to solve this problem. Figure 31 describes the mean location error reduces as the batch size increases. The best localization performance of 191.52 [m] was achieved when the batch size was 500. To make a note, increasing batch-size is good for achieving training accuracy but it takes more computational cost to train the model [24].

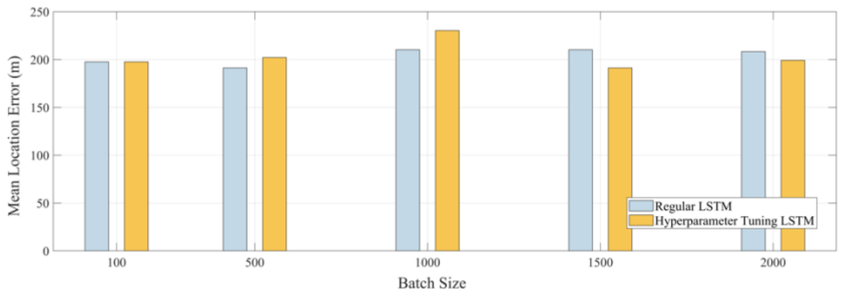


Figure 31: Impact of batch size on mean location error

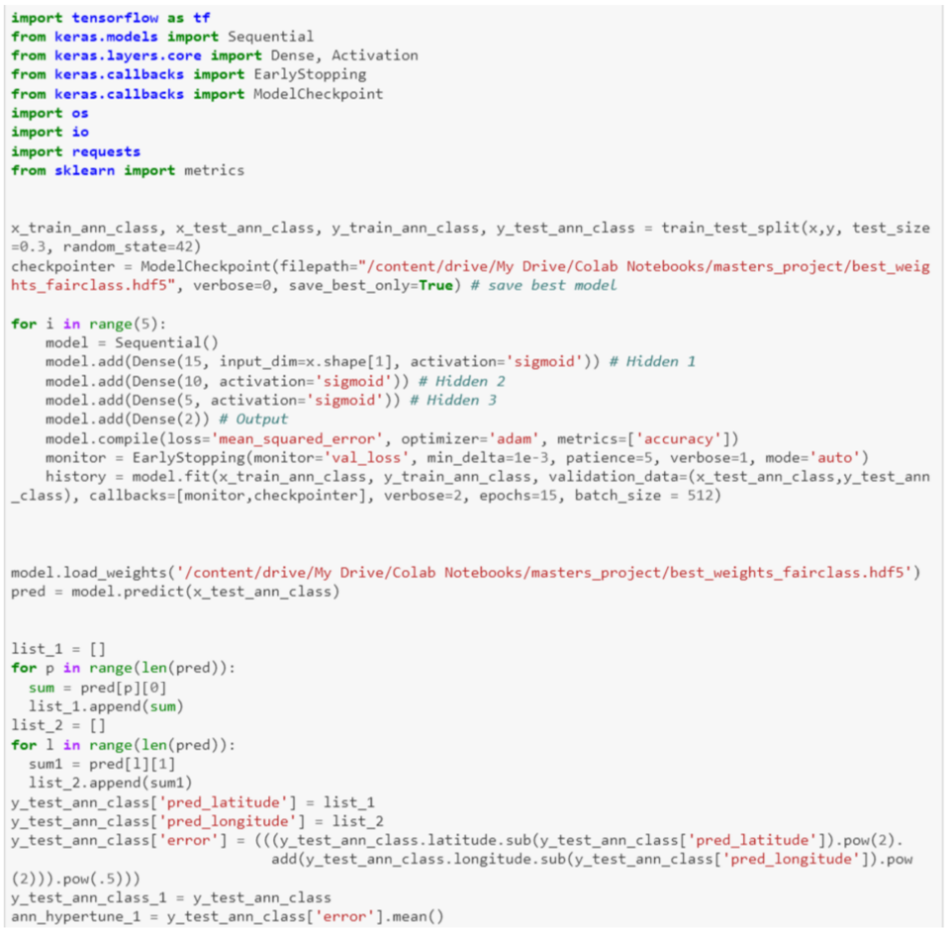
5.9 Analysis of Experimental Indoor Localization Results

Using machine-learning models like *k*-NN and support vector regression, one can estimate the node-location by measuring and determining the fingerprint distance at the unknown point in the database using the nearest neighbors' concept [26].

The indoor data collected using the setup discussed above was split into a 70:30 ratio. The basic configuration such as the number of hidden-layers, activation functions, optimizers, number of neurons in each layer, dropout-layer, batch-size, and epoch count were decided based on the experimental setup mentioned in the paper titled 'Recurrent Neural Networks for Accurate RSSI Indoor Localization' [2].

The code snippet for our best model of the artificial neural network is shown in Figure 32. The data after the preprocessing phase is split into training and testing with a train_test_split function in pandas. Model checkpoint is a callback class to save a copy of the model each time the validation score of the neural network improves. It takes file-path as an argument and helps to revert at the point of failure. The complete neural network model will be saved in the HDF5 file format. The 'Sequential' model is created which takes a list of neural network layers information in this constructor. Multiple layers have been added using the add() method. The weights of the neurons are adjusted using sigmoid as an activation function while Adam is used as an optimizer to minimize validation loss. The loss function used here is the mean squared error. The model is configured using model.compile() to start the learning process, before training. These models are built using Keras [13] and can take input data as numpy arrays. The model is trained using model.fit() and predicted using model.predict(). The last part of the code explains how to create a list for predicted and test latitude and longitude after training. The mean distance error is calculated using the Euclidean distance formula in pandas.

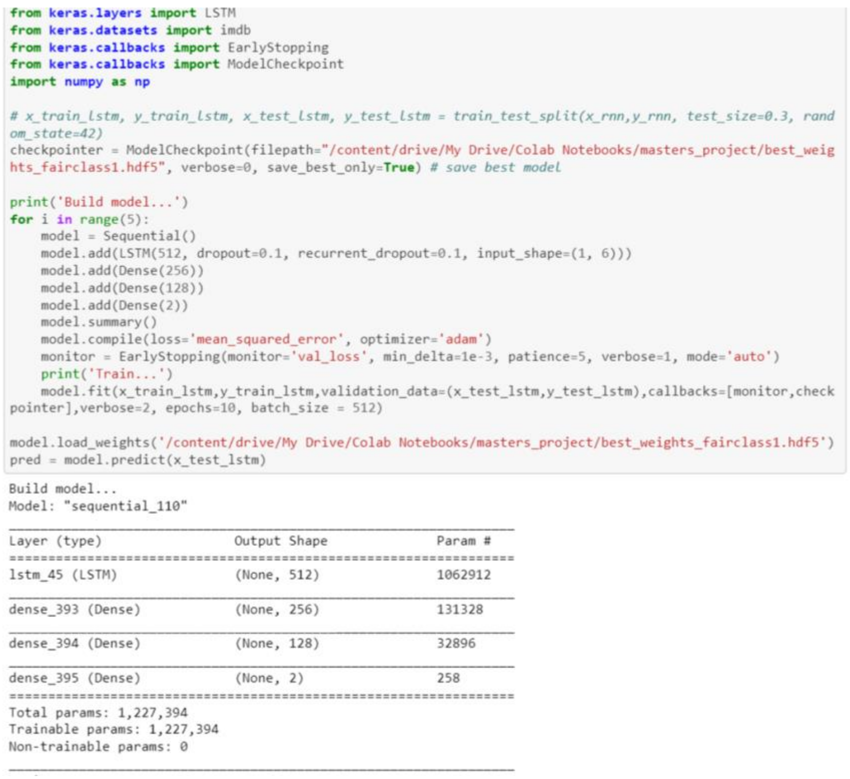
For 1st iteration, the validation loss was 0.9925 and training loss was 1.5743. For 2nd iteration, the validation loss 1.7897 and training loss was 1.5743. After 5th iteration, the validation loss was 1.0503 and training loss was 1.1171, which indicates that the model stopped its training after achieving the best accuracy. Epoch size is set to 15 with a batch size of 512.



```
import tensorflow as tf
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
import os
import io
import requests
from sklearn import metrics


x_train_ann_class, x_test_ann_class, y_train_ann_class, y_test_ann_class = train_test_split(x,y, test_size
=0.3, random_state=42)
checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/Colab Notebooks/masters_project/best_weig
hts_fairclass.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    model = Sequential()
    model.add(Dense(15, input_dim=x.shape[1], activation='sigmoid')) # Hidden 1
    model.add(Dense(10, activation='sigmoid')) # Hidden 2
    model.add(Dense(5, activation='sigmoid')) # Hidden 3
    model.add(Dense(2)) # Output
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    history = model.fit(x_train_ann_class, y_train_ann_class, validation_data=(x_test_ann_class,y_test_ann
_class), callbacks=[monitor,checkpointer], verbose=2, epochs=15, batch_size = 512)



model.load_weights('/content/drive/My Drive/Colab Notebooks/masters_project/best_weights_fairclass.hdf5')
pred = model.predict(x_test_ann_class)


list_1 = []
for p in range(len(pred)):
  sum = pred[p][0]
  list_1.append(sum)
list_2 = []
for l in range(len(pred)):
  sum1 = pred[l][1]
  list_2.append(sum1)
y_test_ann_class['pred_latitude'] = list_1
y_test_ann_class['pred_longitude'] = list_2
y_test_ann_class['error'] = (((y_test_ann_class.latitude.sub(y_test_ann_class['pred_latitude']).pow(2).
                              add(y_test_ann_class.longitude.sub(y_test_ann_class['pred_longitude']).pow
(2))).pow(.5)))
y_test_ann_class_1 = y_test_ann_class
ann_hypertune_1 = y_test_ann_class['error'].mean()
```

Figure 32: Code Snippet for Artificial Neural Network implementation

Our next approach uses sequential LSTM which removes the assumption of Gaussian noise, by collecting the data at different locations in the floor lobby. It adjusts the internal weights through the training stage. The weighted average filter is applied to overcome the signal instability issues during training and test RSSI calculations. The code snippet of how LSTM is implemented is shown in Figure 33. Each sequence has 1 member and each member is 6 input features which are represented by input_shape = (1, 6). The first LSTM layer has 512 neurons with dropout 0.1 and recurrent_dropout set as 0.1. The dropout is applied to input or output data to mask the vertical components, while recurrent dropout masks the horizontal components. Batch size is set to 512 and epoch size is 10. The model summary is also shown in the below figure.



```
from keras.layers import LSTM
from keras.datasets import imdb
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
import numpy as np

# x_train_lstm, y_train_lstm, x_test_lstm, y_test_lstm = train_test_split(x_rnn,y_rnn, test_size=0.3, rand
om_state=42)
checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/Colab Notebooks/masters_project/best_weig
hts_fairclass1.hdf5", verbose=0, save_best_only=True) # save best model

print('Build model...')
for i in range(5):
    model = Sequential()
    model.add(LSTM(512, dropout=0.1, recurrent_dropout=0.1, input_shape=(1, 6)))
    model.add(Dense(256))
    model.add(Dense(128))
    model.add(Dense(2))
    model.summary()
    model.compile(loss='mean_squared_error', optimizer='adam')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    print('Train...')
    model.fit(x_train_lstm,y_train_lstm,validation_data=(x_test_lstm,y_test_lstm),callbacks=[monitor,check
pointer],verbose=2, epochs=10, batch_size = 512)

model.load_weights('/content/drive/My Drive/Colab Notebooks/masters_project/best_weights_fairclass1.hdf5')
pred = model.predict(x_test_lstm)
```

```
Build model...
Model: "sequential_110"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_45 (LSTM)               (None, 512)               1062912   
_________________________________________________________________
dense_393 (Dense)            (None, 256)               131328    
_________________________________________________________________
dense_394 (Dense)            (None, 128)               32896     
_________________________________________________________________
dense_395 (Dense)            (None, 2)                 258       
=================================================================
Total params: 1,227,394
Trainable params: 1,227,394
Non-trainable params: 0
_________________________________________________________________
```

Figure 33: Code Snippet for Long short-term memory implementation

After 5th iteration, training loss was 0.1199 while validation loss was 1.3114 which gives an idea about overfitting and so dropout was introduced as a part of hyperparameter tuning to improve the accuracy. The mean location error of 1.34 [m] was observed using LSTM. The Convolutional neural network requires x_array and y_array as a matrix before passing it to the model for training. The matrix once created is reshaped into the required format of the expected model. The DeepML results for indoor localization are shown in Table 4.

Table 4: Indoor Localization: Mean Location Error [m] results using DeepML

Deep ML Method	**Nodes**	**Epoch**	**Batch Size**	**Dropout**	**Mean Location Error [m]**
ANN	100	10	256	None	1.324271
	30	15	512	None	1.270332
	100	20	256	None	1.286759
LSTM	64	10	512	0.1	1.409174
	128	10	81	0.5	1.348190
	512	20	256	0.3	1.799690
CNN	1024	3	64	0.5	1.804363
	512	3	81	0.5	1.886141
	128	3	128	0.5	1.786397

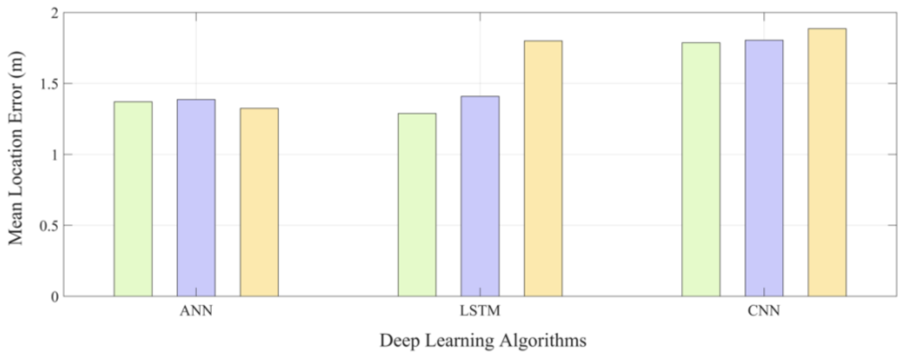


Figure 34: Deep Learning Algorithms Results for Indoor Localization

The cumulative distribution function (CDF) is a statistical function for a random variable X, evaluated at some value y, is the probability that random variable X will take values less than or equal to y [25]. 59% of ANN has a mean distance error of 1.5 [m] range or less, while 28% of CNN and 42% of LSTM have a mean error below of 1.5 [m] or less, observed from CDF in comparison to 38% from the CNN algorithm, plotted in Figure 35 using Matlab [19] ecdf().

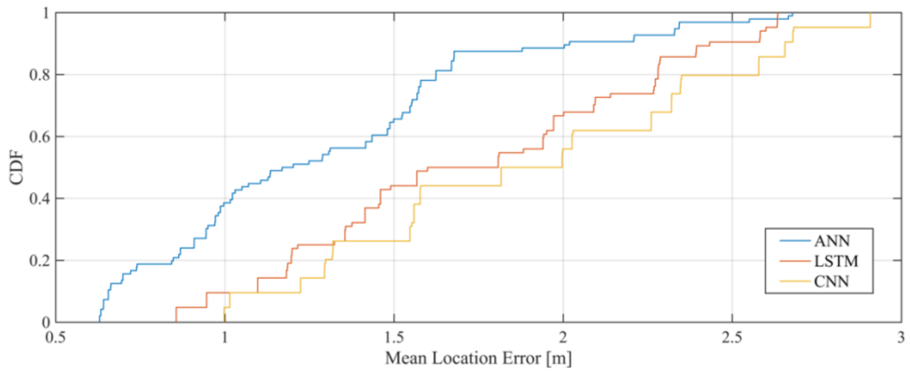


Figure 35: CDF vs Mean Distance Error [m] for Indoor Localization

5.10 Impact of design parameters on LSTM for Indoor Localization

We evaluated the proposed LSTM models with test data size of 25, 50, 75, 100 and 125, to verify how varying test data size impacts the mean location error of indoor localization on LSTM [24]. The epoch, hidden units, optimizer, activation function, batch size, and optimizers are set to 128, 3, Adam, RmsProp, 81 and 0.1 respectively.

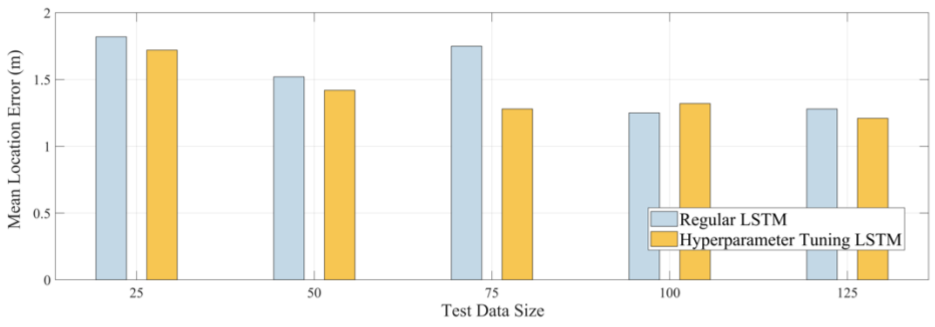


Figure 36: Impact of test data size on mean location error

As shown in Figure 36, distance errors of both scenarios are almost the same for different test data sizes. Specifically, the mean distance errors for regular LSTM and hyper-tuned LSTM are 1.75 [m] and 1.34 [m], respectively, for test data size of 75. Adding more data samples and training locations could help us achieve higher precision and reduce overfitting [24].

Batch size plays an important role in optimizing the model's accuracy. Figure 37 describes the mean location error reduces as the batch size increases. The best localization performance of 1.28 [m] for indoor localization, is achieved when the batch size is 500.

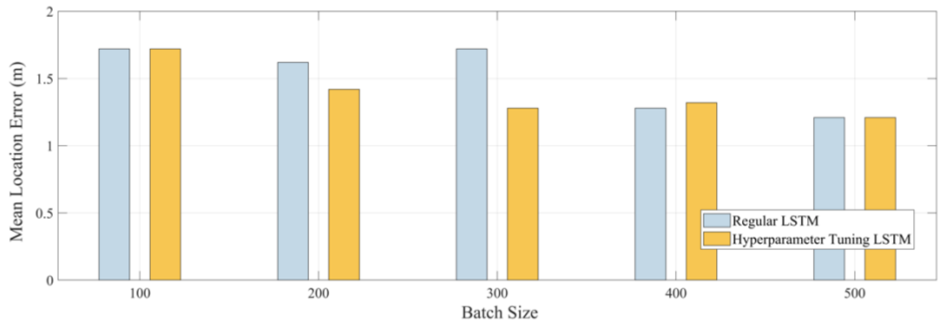


Figure 37: Impact of batch size on mean location error

CHAPTER 6

PROJECT CONCLUSION

This project has described the challenges in fingerprinting-based localization in an indoor and outdoor environment. The results from outdoor localization achieved mean localization error of 191.52 [m] using LSTM structure and out-performs kNN results described in [1]. These algorithms were implemented to improve previously defined results and to analyze short and long-term fluctuations in signal strengths over a large area in the city. This has helped to understand capabilities of LoRa as technology in comparison to Bluetooth and Wi-Fi. This project also described a novel approach for fingerprinting based localization using LoRa for indoor environments based on neural networks. The idea was to analyze indoor localization problems like spatial ambiguity, signal-strength fluctuations and non-linear relations in a typical indoor infrastructure layout and how LoRa and neural networks can help us mitigate them. Experimental results with a specific Dragino LoRa gateway-node set up in a university building have demonstrated that ANN and LSTM achieved 1.27 [m] and 1.34 [m] of mean distance error, which outperforms CNN.

As, the last chapter of this report, I will discuss what are my learnings and what could be the future scope of this application.

6.1 Learnings

Before starting this project, I was only aware of the necessary details about Neural networks and the Internet of Things (IoT) as an application. This project allowed me to explore my knowledge and learn what LoRa technology is? What is the LoRaWAN protocol? What are the challenges in fingerprinting-based localization? How can we handle big data from sensors? How do Cloud services work while working on distributed data? How neural networks train big data? Which data structures are used to store the sensor data? Finding all the nuances for these questions, which are mentioned in previous chapters, I became cognizant about every minute detail of the fingerprinting-based localization using LoRa and neural networks. From the implementation perspective, I learned how to setup LoRa sensor-gateway connection over The Things Network as a cloud service, how to work on huge data and preprocess it understanding every minute detail of its domain, training and evaluating neural network models, its architecture, and impact of hyperparameter tuning on the model's accuracy.

6.2 Future Work

- Implemented distributed, scaled end-to-end cloud RESTful API framework,
- Federated Machine Learning on the edge to solve localization challenges,
- Try other deep-learning models like bi-directional LSTM, GRU,
- Use naïve bayes theorem for estimating probability of base-stations,
- Treat Localization as a classification problem instead of regression,
- Security Implementation of LoRaWAN in indoor localization.

Bibliography

[1] M. Aernouts, R. Berkvens, W. Maarten, and K.V. Vlaenderen, "Sigfox and LoRaWAN Datasets for Fingerprint Localization in Large Urban and Rural Areas," *MDPI*, vol. 3, no. 2, 2018.

[2] M. Hoang, B. Yuen, X. Dong, T. Lu, R. Westendorp and K. Reddy, "Recurrent Neural Networks For Accurate RSSI Indoor Localization," *ResearchGate*, March 2019.

[3] B. Islam, M. T. Islam, J. Kaur and S. Nirjon, "LoRaIn: Making a Case for LoRa in Indoor Localization," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kyoto, Japan, 2019.

[4] D. Xuan, "CSE5432: Mobile Handset Systems and Networking," 2018. [Online]. Available: http://www.cse.ohio-state.edu/~xuan/courses/5432/5432_localization.ppt . [Accessed September 2019].

[5] C. Lawton, "Connect your devices to AWS IoT using LoRaWAN," 2018. [Online]. Available: https://aws.amazon.com/blogs/iot/connect-your-devices-to-aws-iot-using -lorawan/. [Accessed September 2019].

[6] Admin, "LoRa Alliance: LoRa Specification Documentation," [Online]. Available: https://lora-alliance.org/about-lorawan. [Accessed September 2019].

[7] "Difference between Neural Networks and Deep Learning," [Online]. Available: https://www.educba.com/neural-networks-vs-deep-learning/. [Accessed September 2019].

[8] Wikipedia, "Deep Learning," [Online]. Available: https://en.wikipedia.org/wiki /Deep_learning. [Accessed September 2019].

[9] C. Olah, "Understanding LSTM Networks - Colah's Blog," August 2017. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed October 2019].

[10] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks – The ELI5 way," December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed October 2019].

[11] J. Rodriguez, "Understanding Hyperparameters Optimization in Deep Learning Models: Concepts and Tools," August 2018. [Online]. Available: https://towardsdatascience.com/understanding-hyperparameters-optimization-in-deep-learning-models-concepts-and-tools-357002a3338a. [Accessed October 2019].

[12] "TensorFlow API Documentation," Google, [Online]. Available: https://www.tensorflow.org/api_docs/. [Accessed October 2019].

[13] "Keras: The Python Deep Learning Library," [Online]. Available: https://keras.io/models/about-keras-models/. [Accessed October 2019].

[14] "Supervised Neural Networks – scikit-learn documentation," [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Accessed October 2019].

[15] "Getting Started with Google Colab," [Online]. Available: https://towardsdatascience .com/getting-started-with-google-colab-f2fff97f594c. [Accessed October 2019].

[16] "Single Channel LoRa IoT Kit v2 User Manual," Dragino Technology Company, [Online]. Available: https://www.dragino.com/downloads/index.php?dir=LoRa_IoT _Kit/v2-Kit/. [Accessed October 2019].

[17] "Python 3.7.5 documentation," [Online]. Available: https://docs.python.org/3.7/. [Accessed October 2019].

[18] "C Language Programming," [Online]. Available: https://www.geeksforgeeks.org/c-language-set-1-introduction/. [Accessed October 2019].

[19] "ECDF using Matlab," Matlab, [Online]. Available: https://www.mathworks.com /help/stats/ecdf.html. [Accessed November 2019].

[20] "Pandas 0.25.1 Documentation: powerful Python data analysis toolkit," [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/. [Accessed October 2019].

[21] Sanat, "Data Visualization using Python for Machine Learning and Data Science," December 2018. [Online]. Available: https://towardsdatascience.com/data-visualization-for-machine-learning-and-data-science-a45178970be7. [Accessed October 2019].

[22] V. Valkov, "Data Imputation using Autoencoders | What to do when data is missing?," [Online]. Available: https://www.curiousily.com/posts/data-imputation-using-autoencoders/. [Accessed October 2019].

[23] Wikipedia, "Linear Interpolation," [Online]. Available: https://en.wikipedia.org/ wiki/Linear_interpolation. [Accessed November 2019].

[24] X. Wang, Z. Yu and S. Mao, "DeepML: Deep LSTM for Indoor Localization with Smartphone Magnetic and Light Sensors," in *2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, 2018.

[25] Wikipedia, "Cumulative Distribution Function," [Online]. Available: https://en. wikipedia.org/wiki/Cumulative_distribution_function. [Accessed November 2019].

[26] X. Wang, Z. Yu and S. Mao, "CSI-Based Fingerprinting for Indoor Localization: A Deep Learning Approach," *in IEEE Transactions on Vehicular Technology,* vol. 66, no. 1, pp. 763-776, Jan. 2017.