A Comparison of Deep Learning Models for IQ Fingerprint Map Based Indoor Positioning in Ship Environments

Yootae Shin[1], Qianfeng Lin[2], and Jooyoung Son[3]*
[1]Division of Marine Information Technology, Korea Maritime and Ocean University
Busan, Republic of Korea
[e-mail: sytvip1@gmail.com]
[2]Department of Computer Engineering, Korea Maritime and Ocean University
Busan, Republic of Korea
[e-mail: linqianfeng@g.kmou.ac.kr]
[3]Division of Marine IT Engineering, Korea Maritime and Ocean University
Busan, Republic of Korea
[e-mail: mmlab@kmou.ac.kr]
*Corresponding author: Jooyoung Son

Received January 5, 2024; revised March 14, 2024; accepted March 23, 2024; published April 30, 2024

Abstract

The importance of indoor positioning has grown in numerous application areas such as emergency response, logistics, and industrial automation. In ships, indoor positioning is also needed to provide services to passengers on board. Due to the complex structure and dynamic nature of ship environments, conventional positioning techniques have limitations in providing accurate positions. Compared to other indoor positioning technologies, Bluetooth 5.1-based indoor positioning technology is highly suitable for ship environments. Bluetooth 5.1 attains centimeter-level positioning accuracy by collecting In-phase and Quadrature (IQ) samples from wireless signals. However, distorted IQ samples can lead to significant errors in the final estimated position. Therefore, we propose an indoor positioning method for ships that utilizes a Deep Neural Network (DNN) combined with IQ fingerprint maps to overcome the challenges associated with accurate location detection within the ship. The results indicate that the accuracy of our proposed method can reach up to 97.76%.

Keywords: Indoor positioning, Deep learning, IQ fingerprint map, Bluetooth 5.1, ship environments

---

A preliminary version of this paper was presented at APIC-IST 2023, and was selected as an outstanding paper.

http://doi.org/10.3837/tiis.2024.04.017 **ISSN : 1976-7277**

1. Introduction

According to a report dated February 11, 2024, an illness outbreak aboard a Cunard Cruise Line ship has spread, now leaving at least 154 people experiencing symptoms such as vomiting and diarrhea, according to the U.S. Centers for Disease Control and Prevention [1]. Even though the COVID-19 pandemic has subsided, other viruses still pose a risk of causing widespread transmission in enclosed environments like cruise ships. In such settings, the speed and scope of virus transmission can be heightened. Therefore, timely isolation of close contacts onboard becomes particularly crucial to prevent rapid virus spread. However, traditional methods of identifying close contacts primarily rely on inquiries, which are not only inefficient but also prone to significant errors. To address this issue, a novel approach has been proposed in the literature [2], which involves tracking close contacts through Bluetooth signals. The core idea of this method lies in utilizing Bluetooth signals to estimate the positions of passengers on the ship and further analyzing the position to identify close contacts. A Close Contact Identification Algorithm (CCIA) suitable for ship environments is introduced in the literature [3]. The key concept of CCIA is to identify close contacts by computing the probability density of position. Therefore, the primary issue to address is indoor positioning on ships, which is also the focus of this paper.

Indoor positioning has become increasingly important in various application domains, including emergency response, logistics, and industrial automation. In the domain of Indoor Positioning Systems (IPS), a variety of navigation technologies including accelerometers, gyroscopes, compasses, along with Radio Frequency Identification (RFID), Wi-Fi (Wireless Fidelity), and Bluetooth, play pivotal roles. These technologies are harnessed to emit positioning signals, which, when integrated with sophisticated algorithms, enable precise position tracking. Among these technologies, Bluetooth distinguishes itself by its notably lower energy consumption, allowing devices to transmit data over extended periods without frequent recharging. This is particularly advantageous over Wi-Fi, which is known for its substantial power demands, rendering it less suitable for battery-operated devices in ship environments where energy efficiency is paramount. Consequently, Wi-Fi-based tracking is often deemed less favorable for use in ship environments.

Moreover, when comparing costs and operational range, Bluetooth emerges as more cost-effective than Ultra-Wideband (UWB) and boasts a greater range than ZigBee, making it a superior choice in certain scenarios. RFID-based positioning systems necessitate a setup of tags, readers, and antennas, adding to the complexity and cost. In contrast, the advent of Bluetooth 5.1 technology simplifies this by only requiring tags and antennas for indoor positioning, offering a streamlined and efficient solution ideal for ship environments. Thus, Bluetooth 5.1 stands out as a promising approach for achieving accurate indoor positioning in ship environments, balancing both power efficiency and range. Bluetooth Low Energy (BLE) tags can be attached to user devices, such as smartphones, to provide accurate indoor positioning [4].

In wireless communication systems, the signal is often represented as a complex number in the form of In-phase (I) and Quadrature (Q) components, collectively known as In-phase and Quadrature (IQ) samples. Bluetooth 5.1 achieves centimeter-level positioning by collecting IQ samples of wireless signals. In Bluetooth 5.1 technology, IQ samples play a crucial role in accurately estimating the Angle of Arrival (AoA) for positioning [5]. Therefore, we create IQ fingerprint maps from the IQ samples collected at reference points (RPs) and combine them with Deep Neural Networks (DNNs) to address the indoor positioning problem on ships.

2. Related Works

Traditional Bluetooth indoor positioning technology mainly relies on the Received Signal Strength Indicator (RSSI) between BLE beacons and receiving devices (such as smartphones) to estimate distances, thereby achieving indoor positioning [6]. Although widely used in various applications, traditional Bluetooth indoor positioning technology still has some drawbacks and limitations. Various factors in indoor environments, such as walls, doors, human bodies, and other obstacles, can interfere with Bluetooth signals, leading to fluctuations in RSSI that affect the accuracy of distance estimation and thus the overall precision of the positioning system. In indoor environments, Bluetooth signals may reflect, refract, or scatter, causing the same beacon signal to reach the receiver through multiple paths. This multipath effect introduces uncertainty in received signal strength, increasing positioning errors [7].

Bluetooth 5.1 introduces Direction Finding functionality, including angle estimation techniques such as AoA. Utilizing IQ samples for positioning is a key component of AoA technology, which brings significant advantages to indoor positioning [8]. By leveraging IQ samples, AoA positioning technology can accurately measure the angle of signal arrival. This precise angle measurement enables the positioning system to calculate device positions more accurately, thereby greatly improving positioning accuracy. However, in ship environments, IQ samples are highly susceptible to distortion.

The complex structure and dynamic nature of ship environments pose significant challenges to existing indoor positioning techniques. Due to the metallic environment of ships, multipath effects may occur. Multipath propagation is a significant issue in metallic environments like ships. When wireless signals encounter metal surfaces, they can be reflected, resulting in multiple copies of the transmitted signal at the receiver. These reflected signals can interfere with the original signal, leading to constructive or destructive interference, which can distort the received IQ samples and affect the overall performance of indoor positioning systems. In ship environments, the transmitted wireless signal can follow multiple paths due to reflection, diffraction, and scattering caused by the metallic structures. As a result, multiple delayed copies of the signal reach the receiver, which can cause inter-symbol interference and distort the received IQ samples. Distorted IQ samples can considerably affect the performance of indoor positioning systems, especially those that rely on AoA measurements, such as Bluetooth 5.1 technology. Utilizing severely distorted IQ samples for position estimation is not capable of achieving centimeter-level positioning accuracy.

IQ samples can be used to estimate elevation and azimuth. Elevation and azimuth can be used to estimate position, which is the AoA localization method. Hence, the accuracy of elevation and azimuth becomes particularly crucial. Nonetheless, due to the influence of multipath effects, the estimation results of elevation and azimuth are often not precise enough. Thus, in the literature [9], a Mean Optimization Filter (MOF) is proposed to enhance the accuracy of the Bluetooth 5.1 AoA indoor positioning technology. The concept of MOF is to mitigate the impact of noisy data by identifying the optimal average within the dataset. Nevertheless, due to the continual changes in the surrounding environment, MOF cannot autonomously adjust relevant parameters from the data. Consequently, [10] introduces a Self-Learning Mean Optimization Filter (SLMOF). SLMOF can automatically adjust relevant parameters based on real-time data to ensure the accuracy of the optimal average. However, since the AoA localization method requires estimating elevation and azimuth first using IQ samples, and then estimating position based on elevation and azimuth, this leads to further accumulation and amplification of errors. Therefore, our idea is to directly estimate position using IQ samples. There exists a complex mathematical relationship between IQ samples and positions.

Deep Neural Network (DNN), compared to traditional shallow neural networks, has a deeper structure, enabling it to learn more hierarchical and abstract feature representations, thus better capturing the complex relationships within data. Traditional neural networks require manual feature extractors, while DNN can directly learn features from raw data through end-to-end learning, avoiding the cumbersome process of manual feature engineering [11].

Convolutional Neural Networks (CNNs) are primarily used for processing image data, and their ability to handle structured data (such as IQ samples) is relatively weak, whereas DNNs are better suited for handling structured data and can better learn the nonlinear relationships between data [12]. CNNs have a local perception in feature extraction, while DNNs can learn more abstract and global features at different levels, making them more suitable for handling data with complex feature relationships. Recurrent Neural Networks (RNNs) suffer from the problem of vanishing or exploding gradients when dealing with long sequences, whereas DNNs are relatively easier to train and optimize [13]. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), because of their focus on handling data with temporal dependencies, perform well in fields such as language modeling, text generation, speech recognition, and time series prediction [14]. For fixed-length data points, such as the mapping relationship between IQ samples and positions, DNNs can directly learn decisive features from the data without considering temporal dependencies, making them more suitable for such tasks. In some cases, if the dataset lacks strong temporal characteristics, using LSTM or GRU may introduce unnecessary complexity without significant performance improvement.

In summary, compared to traditional neural networks, CNNs, RNNs, LSTM, and GRU, DNNs have deeper structures, better feature learning capabilities, are more suitable for handling structured data, and are easier to optimize [15]. DNN exhibits significant robustness when handling the mathematical relationship between IQ samples and positions, especially in complex environments such as ship environments with metallic structures. In ship environments, metallic structures can induce signal reflection and scattering, further exacerbating multipath effects, which significantly affect the accuracy of localization methods based on traditional approaches. However, DNN, with its deep structure and sophisticated data processing capabilities, can effectively identify and learn the specific impact of these environmental factors on IQ samples. Therefore, they are more suitable for finding the mathematical relationship between IQ samples and positions.

The role of the optimizer in a DNN is to minimize the loss function by adjusting the internal parameters of the network, such as weights and biases. The goal of a DNN is to learn the mapping from input data to output data for a specific task, such as classification or regression. During training, the optimizer iteratively adjusts the network parameters to make the network's output as close as possible to the ground truth, thereby reducing prediction error or loss.

DNNs require optimizers because finding the optimal parameter combination directly is a complex and time-consuming task, especially considering the large number of parameters in a typical DNN. Optimizers employ various optimization algorithms, such as gradient descent and its variants, to automatically adjust parameters and minimize the loss function.

Adam, RMSprop, Stochastic Gradient Descent (SGD), Adagrad, and Adamax are optimization algorithms commonly used in training DNNs [16]. Adam combines the benefits of RMSprop and momentum optimization, adjusting learning rates adaptively for each parameter. RMSprop adjusts learning rates based on the average of recent gradients, while SGD updates parameters using gradients of the loss function [17]. Adagrad adapts learning rates based on historical gradient information, and Adamax is a variant of Adam that uses the infinity norm for adaptive learning rates. Each optimizer has its strengths and weaknesses, and the choice depends on factors like dataset size, network architecture, and computational resources. Experimentation with different optimizers and hyperparameters is often necessary to find the most effective optimization strategy for a given task. This paper aims to address these challenges by proposing a method of using IQ fingerprint maps based on DNN in ship environments. We determine the appropriate optimizer through multiple experiments.

The originality of our work lies in using different neuron configurations to directly handle the complex relationship between IQ samples and position estimation. Through empirical studies of different model architectures, we demonstrate the effectiveness and adaptability of DNN in addressing this specific application.

3. DNN for Bluetooth 5.1 Indoor Positioning with IQ Fingerprint Map

3.1 Acquisition Method of IQ Samples

Commonly, the RSSI fingerprint map serves as a prevalent technique for indoor positioning [18]. However, RSSI is vulnerable to multipath effects, which may result in signals reflecting off objects and traversing multiple routes before reaching the receiver [19]. Generally, Bluetooth RSSI values span from -26 to -100 as integers. Consequently, it becomes challenging to accurately depict wireless signal characteristics using RSSI alone.
In products developed by Nordic Semiconductor, out-of-range IQ samples are capped at an integer value of -32768, signifying that the value range for IQ samples is considerably broader than that of the RSSI. According to the Bluetooth 5.1 Core Specification, the Constant Tone Extension (CTE) of a packet comprises a 4 μs guard period, an 8 μs reference period, and a specific number of antenna samples [20]. When a received packet contains an AoA CTE, the receiver must execute IQ sampling based on the rate and pattern established by the host. **Fig. 1** illustrates how to obtain IQ samples. The BLE tag sends Bluetooth signals to the surroundings, and upon receiving the Bluetooth signal, the AoA locator executes IQ sampling based on CTE. IQ represents a pair of data. I_n refers to the "I" component in the n_{th} set of IQ data. Q_n refers to the "Q" component in the n_{th} set of IQ data.

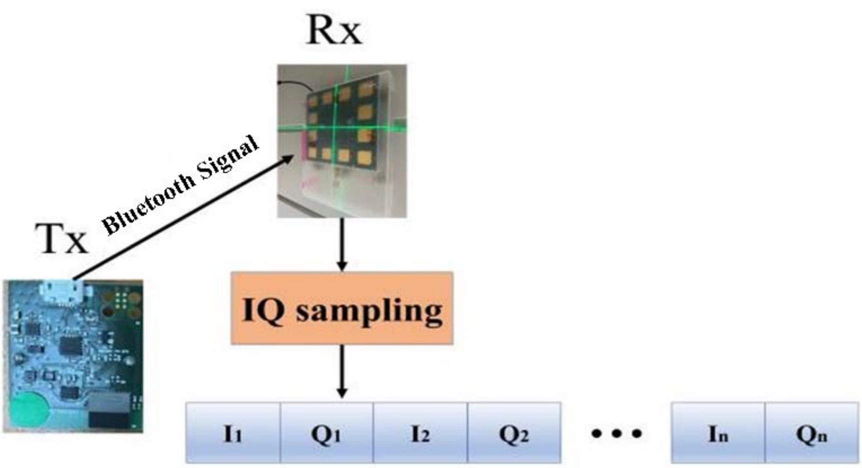


Fig. 1. The AoA locator (Rx) receives Bluetooth signals from the BLE tag (Tx) and performs IQ sampling

In ship environments, dealing with distorted IQ samples can be challenging. The main reasons for choosing DNN over other methods for handling distorted IQ samples are that DNNs are capable of automatically identifying and learning complex features from large-scale data, dealing with non-linear problems, and they possess strong generalization capabilities to handle unseen new samples. Moreover, their multi-layer architecture allows understanding data at different levels of abstraction. These advantages make DNNs more efficient and accurate for processing complex and distorted IQ samples compared to other methods. In this paper, a well-trained DNN with optimizer can be employed to classify the distorted IQ samples effectively.

3.2 Adagrad optimizer in DNN

Adagrad is an adaptive gradient descent algorithm designed to adjust the learning rate for each parameter. The core idea is to adapt the learning rate based on the frequency of parameter updates, using a smaller learning rate for frequently updated parameters and a larger learning rate for infrequently updated parameters. This differential adjustment makes Adagrad particularly suitable for handling sparse data [21].

In Adagrad, the algorithm first computes the gradient for each parameter and then adds the square of this gradient to an accumulated sum. This accumulated sum is subsequently used to adjust the learning rate for that parameter. Specifically, the learning rate is adjusted by dividing the global learning rate by the square root of the accumulated sum. Consequently, parameters with larger accumulated gradient sums have smaller learning rates, while those with smaller sums have larger learning rates.

The adaptive learning rate mechanism of Adagrad reduces the need for manual adjustment of learning rates, and is able to automatically find a relatively reasonable learning rate across different stages and frequencies of parameter updates.

G_t is a diagonal matrix, where each diagonal element $G_{t,ii}$ represents the sum of squares of gradients for the i_{th} parameter up to time step t. The core of the Adagrad algorithm is to maintain an accumulated square gradient $G_{t,ii}$ for each parameter, which is the sum of squares of gradients up to the current time step. For each iteration, $G_{t,ii}$ is updated as Eq. 1 [21].

$$G_{t,ii} = G_{t-1,ii} + g_{t,i}^2 \tag{1}$$

$G_{t,ii}$ starts with an initial value of 0 and gradually increases with each parameter update. $g_{t,i}$ represents the gradient of the i_{th} parameter at time step t. When updating parameters, Adagrad adjusts the learning rate for each parameter according to the following Eq. 2.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} g_{t,i} \tag{2}$$

η is the global learning rate, and ϵ is a very small constant to avoid division by zero. In this way, the update step size for each parameter is adaptively adjusted, depending on its own historical gradient information.

Through the mechanism described above, Adagrad assigns an adaptive learning rate to each parameter based on its historical gradient information. The learning rate for parameters gradually decreases as their gradient accumulates. This means that for parameters that are frequently updated, their learning rates are relatively low to avoid excessive updates caused by large step sizes. Conversely, for parameters that are infrequently updated, their learning rates are be relatively high due to their smaller accumulated gradient squares, thus promoting faster learning for these parameters. The adaptive learning rate feature of Adagrad makes it particularly effective in handling datasets with sparse gradients, as it can provide larger update steps for parameters that are only occasionally updated, accelerating the learning process. Adagrad's role in DNNs is to adaptively adjust the learning rate for each parameter based on its gradient's historical information. This method effectively handles sparse data and non-stationary objective functions, making the training process more stable and efficient.

3.3 SGD optimizer in DNN

SGD can assist DNN in effectively distinguishing distorted IQ samples because it optimizes models efficiently, handles large-scale data, and mitigates overfitting. This enables the DNN to learn key features to identify distorted IQ samples within complex, non-linear, and large-scale datasets. Moreover, the randomness of SGD increases the diversity of the training process, enhancing the generalization capabilities of the model to better deal with new, unseen distorted samples. SGD can help DNNs achieve desirable performance. SGD is a simple yet efficient optimization algorithm that computes the gradient of the loss function with respect to each weight by using a small subset of the training data. The weight update as Eq. 3 [22].

$$w = w - lr \times \frac{dL}{dW} \tag{3}$$

W represents the weights, *lr* is the learning rate, and *dL/dW* is the gradient of the loss function with respect to *W*. Ship environments are often characterized by high levels of noise, which can cause distortion in IQ samples. The inherent noise in SGD's weight updates allows it to be more robust in the presence of noisy data. SGD, as an optimization algorithm, plays a crucial role in deep learning models, including the DNN. One of the main reasons is its capability to help the model escape local minima during the learning process.

DNN involves finding the minimum value of a loss function in a high-dimensional space, where each point corresponds to a set of model parameters. Within this high-dimensional space, the loss function often has multiple local minima, which might correspond to states where the model overfits specific data or fails to fully learn.

3.4 DNN-based In-Ship Positioning

Distorted IQ samples often contain more complex and harder-to-capture patterns, this makes the landscape of the loss function even more intricate with a higher number of local minima. If a DNN gets stuck in these local minima during training, it may result in a model that can only adapt to specific training samples and performs poorly when predicting new, unseen distorted samples.

For example, the model may find a way to handle a specific type of noise in the training set at a local minimum, which performs well on the training set but fails when dealing with new distorted samples with different noise. In another case, the model may get stuck in a state at a local minimum where it's overly sensitive to specific sample features. This can result in good performance when handling samples containing these features but a decline in prediction ability for new samples without these features. Therefore, an optimizer is needed to reduce the occurrence of such situations.

Fig. 2 illustrates the workflow of utilizing DNN for in-ship positioning. During the training phase, the IQ fingerprint maps are initially fed into a DNN for training, with the positions serving as labels, to enable the DNN to perform classification tasks. Subsequently, an optimizer is used to automatically adjust the relevant parameters of the DNN.

During the practical application phase, upon receiving a localization request, IQ samples are first obtained. These samples are then input into the pre-trained DNN, resulting in the corresponding position results. This process achieves accurate indoor positioning by learning patterns within the IQ fingerprint maps, providing real-time position information for various applications such as ship navigation, asset tracking, and resource management.
Additionally, the adaptability of DNN to different environmental conditions and its generalization ability across various ship layouts and structures make it suitable for diverse ship environments. Furthermore, continuous updates and fine-tuning of the DNN model with newly acquired data can further enhance its accuracy and robustness in-ship positioning tasks.

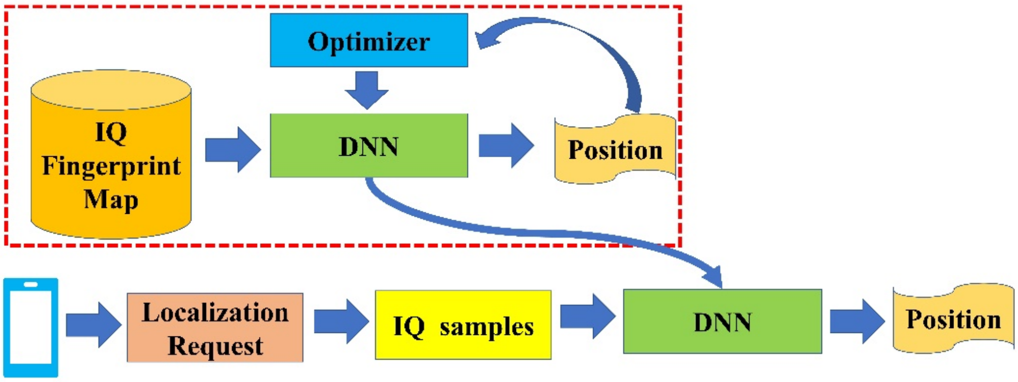


Fig. 2. Workflow of DNN with IQ fingerprint map for estimating position [23]

4. Performance Evaluation

We conduct an indoor positioning experiment to collect IQ samples from a test area situated on the upper deck of the HANNARA training ship used by students. The test area measures 7 meters by 7.14 meters. Within test area, we design12 positions as reference points, as shown in **Fig. 3**.

By optimizing the configuration of hidden layers and nodes, we aim to make the DNN more adaptable to complex data variations. For instance, increasing the depth and number of nodes in the hidden layers can enhance the model's representation capability, allowing it to better capture the data distortion caused by changes in the ship's internal layout. Moreover, appropriately adjusting the structure of hidden layers can make the model more flexible, aiding in its adaptation to different degrees and types of layout changes. The Sigmoid function possesses favorable non-linear properties, performing well when handling non-linear data. When changes occur in the ship's internal layout, the data may exhibit complex non-linear relationships, which the Sigmoid function is better equipped to capture. Consequently, this enhances the model's flexibility and robustness. Additionally, the Sigmoid function's smoothness helps mitigate issues such as gradient vanishing or exploding, thereby improving the stability of model training.

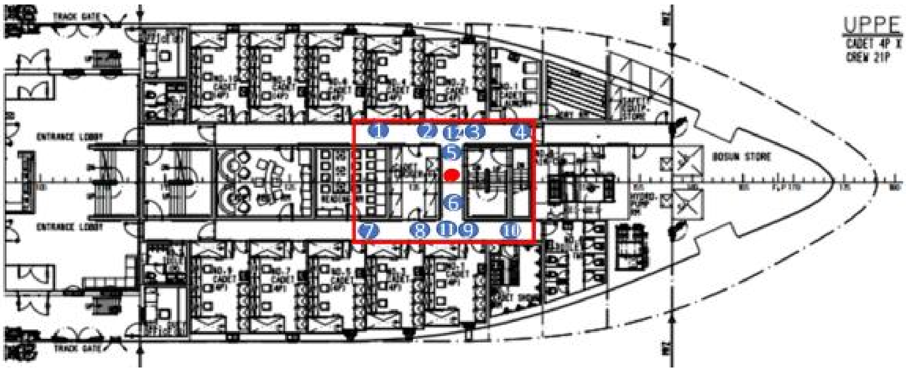


Fig. 3. Floor plan of upper deck in HANNARA ship with reference points

These reference points are designated positions for fingerprint mapping. When IQ samples are input into the DNN, the corresponding reference points comes out as the final output. Totally, the neural network model comprises 44 input features, and 12 output neurons due to the reference points.

The performance of DNN for estimating position is controlled variously by hyperparameters such as learning rate, batch size, and neuron size. But most of hyperparameters are decided by the used optimizer.

The entire training is executed using grid search from the SciKeras package [24]. It combines library Keras used for deep learning and Scikit-Learn used for machine learning, which makes the function of grid search available even in deep learning. It observes the performance differences of models that use difference hyperparameters. The number of epochs is controlled by a callback function, which stops training when the lowest loss function is reached and recorded to measure the computational cost.

Table 1 shows the values of the hyperparameters used in the training. SGD, Adagrad, RMSprop, Adam, and Adamax are used as optimizers. To distinguish between high and low learning rates, the learning rate is trained at 0.01, 0.1, and 1. The batch size is also trained at 32, 64, and 128 for the same reason. Neuron sizes, also known as node sizes, refer to the sizes of hidden layers in the neural network. Two sets of neuron sizes are trained: (480, 240, 60) and (960, 480, 120), both using the Sigmoid function. These different configurations are used to distinguish varying accuracies based on the number of neurons in the layers. The neuron sizes are configured in two different settings for testing, namely (480, 240, 60) and (960, 480, 120). We opt for these two configurations to explore how model complexity affects position accuracy. The larger neuron set (960, 480, 120) aims to provide higher model capacity to capture the complex relationship between IQ samples and positions. The smaller set (480, 240, 60) is used to validate the performance of a lower-complexity model. Through this approach, we can evaluate the impact of model capacity on position accuracy and find the optimal balance between performance and complexity.

Table 1. List of hyperparameters

Hyperparameter	Options
Optimizer	[SGD, Adagrad, RMSprop, Adam, Adamax]
Learning Rate	[0.01, 0.1, 1]
Batch Size	[32, 64, 128]
Neuron Size	[(480, 240, 60), (960, 480, 120)]

Table 2 outlines the layers of a DNN along with the sizes of neurons used in two different tests. The "Layer" column lists the different layers in the network, including the input, hidden, and output layers. The "Neuron Size (Test 1)" and "Neuron Size (Test 2)" columns specify the sizes of neurons in each layer for two separate tests. For example, in the hidden layer, Test 1 has neuron sizes of 960, 480, and 120, while Test 2 has neuron sizes of 480, 240, and 60. The "Activation Function" column indicates the activation function used in each layer, with Sigmoid applied in the hidden layers and Softmax in the output layer.

We choose the Sigmoid function as the activation function for the hidden layers because of its non-linear properties, enabling the model to learn complex decision boundaries. Although the Sigmoid function may lead to the vanishing gradient problem, in our case, we effectively mitigate this issue by carefully selecting the learning rate and other training parameters. Additionally, the smooth gradient characteristics of the Sigmoid function facilitate stable convergence during training. The main reason for using Softmax as the activation function in the output layer is its suitability for multi-class classification tasks and its ability to transform the neural network's output into a probability distribution. The Softmax function converts the raw scores outputted by the neural network into probability values between 0 and 1, where the sum of these probabilities is 1. This is crucial for classification tasks because it allows the model to output the probability of each class, enabling the determination of the most likely class based on these probabilities. Therefore, Softmax is widely used as the activation function in the output layer for classification tasks.

Table 2. Layers of DNN

Layer	Neuron Size (Test 1)	Neuron Size (Test 2)	Activation Function
Input	44	44	
	960	480	
Hidden	480	240	Sigmoid
	120	60	
Output	12	12	Softmax

Fig. 4 represents the neural network architecture of a DNN, illustrating how it processes IQ samples to generate position results. In this diagram, the input layer receives data from IQ samples, which undergo preprocessing and feature extraction before being transmitted to the hidden layers. The hidden layers process the data through a series of neurons and weights, extracting key features and patterns. Subsequently, these features and patterns are passed to the output layer, where the position results are generated based on them. This process is achieved through complex connections between multiple layers and neurons within the DNN model, where the number and arrangement of neurons in each layer can influence the final position results. Thus, this diagram demonstrates how a DNN learns from IQ samples to infer position results, enabling indoor positioning functionality.

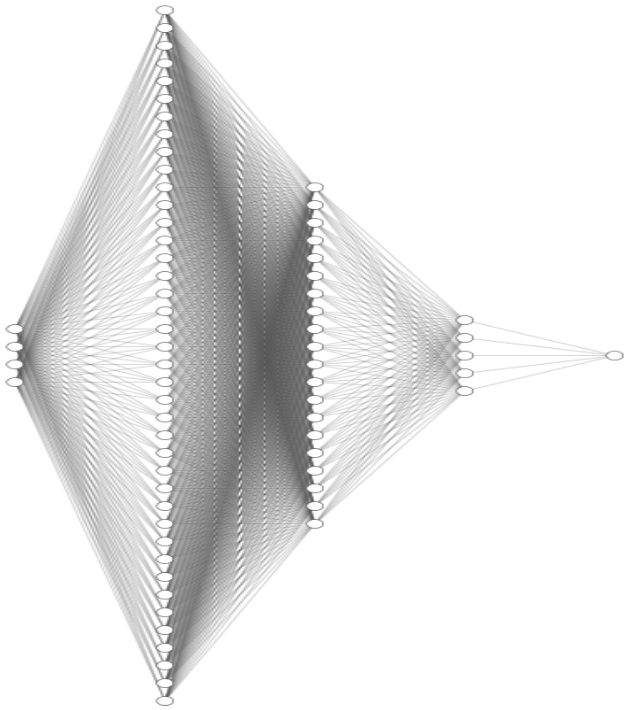


Fig. 4. Deep Neural Network Architecture Diagram: Indoor Positioning Model Based on IQ Samples

Table 3 to 7 displays the training results for SGD optimizer. It lists various training settings including batch size, learning rate, neuron size, and the corresponding accuracy. Each row represents a training experiment where the model is trained multiple times with different parameter settings and evaluated for performance. From the table, it can be observed that with a batch size of 32, a learning rate of 0.1, and larger neuron size ([960 480 120]), the highest accuracy of 0.9776 is achieved. This indicates optimal performance under specific parameter configurations. Furthermore, these results provide valuable insights into the impact of different parameter settings on training outcomes, which can be utilized for adjusting and optimizing neural network models.

Table 3. SGD training results

Index	Batch size	Learning rate	Neuron size	Accuracy
1	32	0.01	[960 480 120]	0.9684
2	32	0.1	[960 480 120]	0.9776
3	32	1	[960 480 120]	0.9721
4	32	0.01	[480 240 60]	0.9592
5	32	0.1	[480 240 60]	0.9675
6	32	1	[480 240 60]	0.9636
7	64	0.01	[960 480 120]	0.9648
8	64	0.1	[960 480 120]	0.9739
9	64	1	[960 480 120]	0.9751
10	64	0.01	[480 240 60]	0.6811
11	64	0.1	[480 240 60]	0.9680
12	64	1	[480 240 60]	0.9705
13	128	0.01	[960 480 120]	0.9519
14	128	0.1	[960 480 120]	0.9711

15	128	1	[960 480 120]	0.9762
16	128	0.01	[480 240 60]	0.1320
17	128	0.1	[480 240 60]	0.9601
18	128	1	[480 240 60]'	0.9708

Table 4 presents the training results of the Adagrad optimizer, with each row representing a training experiment conducted with different parameter configurations. The model's performance is evaluated through multiple training runs. The results indicate that with a batch size of 64, learning rate of 0.1, and neuron size of [960 480 120], the model achieved the highest accuracy of 0.9758, demonstrating relatively stable performance. It is noteworthy that regardless of whether the learning rate is small (0.01) or large (1.0), the model's accuracy remains at a certain level, indicating the stability of Adagrad across different parameter settings.

Table 4. Adagrad training results

Index	Batch size	Learning rate	Neuron size	Accuracy
1	32	0.01	[960 480 120]	0.9708
2	32	0.1	[960 480 120]	0.9738
3	32	1	[960 480 120]	0.9577
4	32	0.01	[480 240 60]	0.9619
5	32	0.1	[480 240 60]	0.9719
6	32	1	[480 240 60]	0.9550
7	64	0.01	[960 480 120]	0.9707
8	64	0.1	[960 480 120]	0.9758
9	64	1	[960 480 120]	0.9662
10	64	0.01	[480 240 60]	0.9599
11	64	0.1	[480 240 60]	0.9706
12	64	1	[480 240 60]	0.9594
13	128	0.01	[960 480 120]	0.9670
14	128	0.1	[960 480 120]	0.9749
15	128	1	[960 480 120]	0.9663
16	128	0.01	[480 240 60]	0.9479
17	128	0.1	[480 240 60]	0.9688
18	128	1	[480 240 60]'	0.9606

Table 5 presents the training results using the RMSprop optimizer. Each row represents a training experiment conducted with different parameter configurations, and the model's performance is evaluated accordingly. From **Table 5**, it can be observed that, with a batch size of 128 and a learning rate of 0.01, using larger neuron sizes [480 240 60] achieved a relatively higher accuracy of 0.9650. However, for other learning rates and neuron sizes, the model's accuracy generally remains low with minimal variation. Overall, RMSprop exhibits unstable performance under these parameter settings and does not demonstrate performance similar to other optimizers.

Table 5. RMSprop training results

Index	Batch size	Learning rate	Neuron size	Accuracy
1	32	0.01	[960 480 120]	0.9359
2	32	0.1	[960 480 120]	0.0899
3	32	1	[960 480 120]	0.0814
4	32	0.01	[480 240 60]	0.9584
5	32	0.1	[480 240 60]	0.0848
6	32	1	[480 240 60]	0.0886
7	64	0.01	[960 480 120]	0.9475
8	64	0.1	[960 480 120]	0.0855
9	64	1	[960 480 120]	0.0803
10	64	0.01	[480 240 60]	0.9619
11	64	0.1	[480 240 60]	0.0867
12	64	1	[480 240 60]	0.0844
13	128	0.01	[960 480 120]	0.9616
14	128	0.1	[960 480 120]	0.0866
15	128	1	[960 480 120]	0.0830
16	128	0.01	[480 240 60]	0.9650
17	128	0.1	[480 240 60]	0.0819
18	128	1	[480 240 60]'	0.0851

Table 6 presents the training results using the Adam optimizer. Each row represents a training experiment conducted with different parameter configurations, evaluating the model's performance. It can be observed from **Table 6** that the model exhibits varying accuracy across different batch sizes, learning rates, and neuron sizes. For instance, with a batch size of 128 and a learning rate of 0.01, relatively higher accuracy is achieved using smaller neuron sizes [480 240 60], reaching 0.9097. However, under other parameter settings, the model generally achieves lower accuracy. Overall, the Adam optimizer demonstrates some instability under these parameter configurations, with a considerable range of accuracy variations.

Table 6. Adam training results

Index	Batch size	Learning rate	Neuron size	Accuracy
1	32	0.01	[960 480 120]	0.0859
2	32	0.1	[960 480 120]	0.0879
3	32	1	[960 480 120]	0.0818
4	32	0.01	[480 240 60]	0.8237
5	32	0.1	[480 240 60]	0.0864
6	32	1	[480 240 60]	0.0851
7	64	0.01	[960 480 120]	0.2771
8	64	0.1	[960 480 120]	0.0813
9	64	1	[960 480 120]	0.0814
10	64	0.01	[480 240 60]	0.8866
11	64	0.1	[480 240 60]	0.0814
12	64	1	[480 240 60]	0.0883
13	128	0.01	[960 480 120]	0.4889
14	128	0.1	[960 480 120]	0.0870
15	128	1	[960 480 120]	0.0860

16	128	0.01	[480 240 60]	0.9097
17	128	0.1	[480 240 60]	0.0790
18	128	1	[480 240 60]	0.0838

Table 7 presents the training results using the Adamax optimizer. Each row corresponds to a training experiment, where multiple training runs are conducted with different parameter configurations, and the model's performance is evaluated based on accuracy. From **Table 7**, it can be observed that the model's accuracy varies across different batch sizes, learning rates, and neuron sizes. For instance, with a batch size of 32 and a learning rate of 0.01, the model achieved relatively high accuracy of 0.9745 when using larger neuron sizes [960 480 120]. However, there is fluctuation in accuracy across other parameter settings, with some cases showing comparatively lower accuracy. Overall, the Adamax optimizer demonstrates a degree of instability in these parameter configurations, with accuracy fluctuating across different settings.

Table 7. Adamax training results

Index	Batch size	Learning rate	Neuron size	Accuracy
1	32	0.01	[960 480 120]	0.9745
2	32	0.1	[960 480 120]	0.0873
3	32	1	[960 480 120]	0.0850
4	32	0.01	[480 240 60]	0.9722
5	32	0.1	[480 240 60]	0.0869
6	32	1	[480 240 60]	0.0899
7	64	0.01	[960 480 120]	0.9715
8	64	0.1	[960 480 120]	0.0900
9	64	1	[960 480 120]	0.0836
10	64	0.01	[480 240 60]	0.9738
11	64	0.1	[480 240 60]	0.0809
12	64	1	[480 240 60]	0.0835
13	128	0.01	[960 480 120]	0.9678
14	128	0.1	[960 480 120]	0.0852
15	128	1	[960 480 120]	0.0820
16	128	0.01	[480 240 60]	0.9700
17	128	0.1	[480 240 60]	0.0854
18	128	1	[480 240 60]	0.0845

Fig. 5 displays the accuracy of different optimizers. The horizontal axis ranges from 0 to 18, representing the index of different experiments, while the vertical axis ranges from 0 to 1, representing accuracy. RMSprop and Adam show significantly different results compared to the SGD and Adagrad. First, RMSprop exhibits higher performance with a learning rate of 0.01, a batch size of 128 and smaller neuron sizes. Similarly, Adam also shows better performance with a learning rate of 0.01, a batch size of 128, and smaller neuron sizes. However, while RMSprop demonstrates stable performance with a learning rate of 0.01, Adam's performance can be quite poor when using a small batch size or larger neuron sizes, even with a learning rate of 0.01. In other words, Adam appears to be highly sensitive to hyperparameters.

Adamax is an extension version of Adam. It modifies the formula applied to the learning rate by using the maximum gradient value instead of accumulating past gradients. However, unlike Adam, besides showing optimal performance with a learning rate of 0.01, Adamax also performs well with 32, a low batch size, and higher neuron sizes, similar to SGD.

Based on the evaluation of different optimizers, we can draw the following conclusions: SGD demonstrates stable and high performance, especially with larger learning rates and smaller batch sizes. Adagrad performs slightly inferior to SGD but still achieves decent performance under certain parameter settings. RMSprop performs well with smaller learning rates and larger batch sizes but exhibits relative instability with considerable performance fluctuations. Adam, similar to RMSprop, shows sensitivity, particularly towards smaller batch sizes and larger neuron sizes, resulting in potentially poorer performance. As an extension of Adam, Adamax demonstrates more stability in certain scenarios, particularly performing well with a learning rate of 0.01. Therefore, selecting the appropriate optimizer should consider factors such as model complexity, data characteristics, and stability requirements, possibly requiring multiple experiments to determine the best choice.

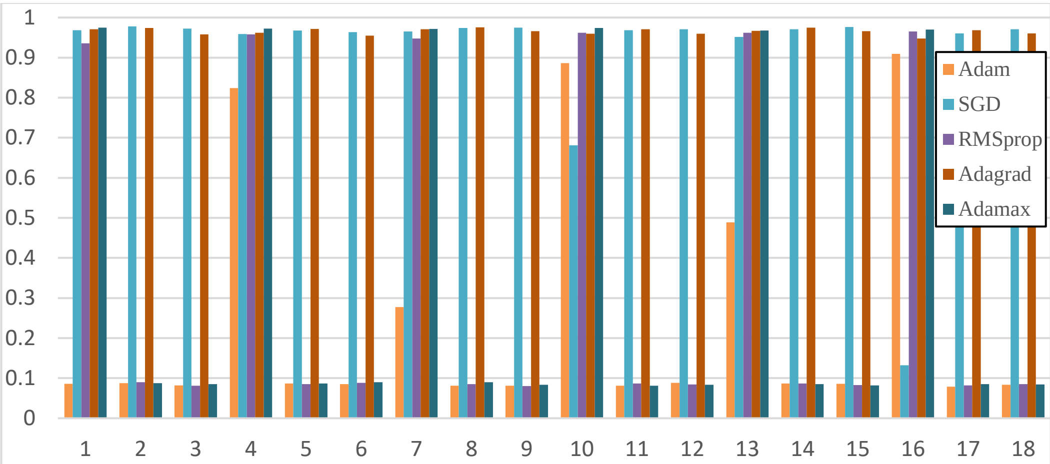


Fig. 5. Accuracy Graph of Optimizers

Table 8 and **Table 9** show hyperparameters based on the best and worst results, respectively, depending on the optimizer used. They are arranged in order from the oldest to the most recent optimizer. One common observation from both tables is that when the training is controlled by a Callback function, which stops training when the loss function does not decrease, there is tendency to show higher accuracy as epochs increase. This suggests that the loss function may be stuck in a local minimum rather than a global minimum when the loss function doesn't decrease. In other words, the hyperparameters shown in each **Table 8** as yielding the best results can be interpreted as the optimal hyperparameters that enable the optimizer to escape from local minima. In **Table 8**, which shows the best cases, we can observe that SGD exhibits the highest accuracy, while Adam shows the lowest accuracy.

Table 8. Best cases of optimizers

Optimizer	Learning rate	Batch Size	Neuron Size	Stop Epochs	Accuracy
SGD	0.1	32	[960, 480, 120]	100	0.977
Adagrad	0.1	64	[960, 480, 120]	89	0.976
RMSprop	0.01	128	[480, 240, 60]	86	0.965
Adam	0.01	128	[480, 240, 60]	75	0.909
Adamax	0.01	32	[960, 480, 120]	93	0.975

On the other hand, in **Table 9**, which displays the worst cases, Adagrad oddly demonstrates a higher accuracy, whereas Adam exhibits the lowest accuracy. Based on these findings, we can conclude that SGD achieves the highest accuracy, Adagrad shows the most stable accuracy, and Adam has the lowest accuracy within the hyperparameters shown in **Table 9**.

Table 9. Worst cases of optimizers

Optimizer	Learning rate	Batch Size	Neuron Size	Stop Epochs	Accuracy
SGD	0.01	64	[480, 240, 60]	45	0.132
Adagrad	0.01	128	[480, 240, 60]	63	0.947
RMSprop	0.1	128	[960, 480, 120]	14	0.08
Adam	0.1	128	[480, 240, 60]	14	0.079
Adamax	0.1	64	[480, 240, 60]	14	0.08

Adam is based on SGD and employs a complex equation. As indicated by the fewest epochs in **Table 8**, it demonstrates excellent efficiency in minimizing the use of computer resources. Nonetheless, due to its complexity, Adam has a lower optimization point and exhibits instability with specific hyperparameter.

We collect IQ samples at 12 reference points and create an IQ fingerprint map. The IQ fingerprint map consists of IQ samples with position indices. We use the IQ fingerprint map as the training set, while the other IQ samples collected at reference points serve as the test set. We input the training set into the DNN and then evaluate the model's performance on the test set. By testing various batch sizes, learning rates, and layer sizes, we eventually obtain the best result. We find that when the batch size is 32, the learning rate is 0.1, and the neuron sizes of layer are [960, 480, 120], the test set's accuracy reaches its highest at 97.76%.

In our concept, IQ samples from Bluetooth 5.1 can be substituted with corresponding data from other localization systems, such as WiFi, RFID, etc., such as Channel State Information (CSI) or RSSI. Therefore, our proposed method can be easily transferred to other localization systems and is not solely dependent on Bluetooth 5.1 technology.

Incremental learning, as a method that enables models to continuously learn from new data and adapt to new circumstances, provides an effective framework for gradually improving positioning performance without the initial need for large amounts of training data [25]. Incremental learning is particularly suited for dynamic environments, such as our application scenario of positioning in ship metal environments, as it allows the model to adapt to subtle environmental changes, such as variations in ship structures, seasonal changes, or other factors that may affect signal characteristics. By continuously integrating newly collected IQ sample data into the training process, the model can automatically enhance accuracy and robustness over time. In the next stage of research, we explore how to integrate incremental learning methods to further investigate the positioning performance of DNN models in ship environments.

5. Conclusions

We evaluate the proposed method in a real ship environment. Firstly, we collect IQ samples at 12 reference points. Secondly, we use the collected IQ samples as the training set and the other collected IQ samples as the test set. Finally, we input the training set into the DNN and evaluate the training performance on the test set. Among the various hyperparameters tested by DNN, we know that the optimizer shows a significant difference in performance. Therefore, various hyperparameters are tested based on the optimizer, and finally, with SGD, we find that when the batch size is 32, the learning rate is 0.1, and the neuron sizes of layer are [960, 480, 120], the test set's accuracy reaches its highest at 97.76%.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (grant number: 2021R1I1A3056125).

References

[1] ABC Chicago, "Illness outbreak spreads on cruise ship, leaving 154 with symptoms such as vomiting, diarrhea,". [Online]. Available: https://abc7chicago.com/cruise-ship-illness-queen-victoria-carnival-2024/14407673, Accessed on: February. 11, 2024.

[2] Q. Lin and J. Son, "A close contact tracing method based on bluetooth signals applicable to ship environments," *KSII Transactions on Internet and Information Systems*, vol. 17, no. 2, pp. 644-662, Feb. 2023. Article (CrossRef Link)

[3] Q. Lin and J. Son, "A close contact identification algorithm using kernel density estimation for the ship passenger health," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 6, pp. 1-14, June. 2023. Article (CrossRef Link)

[4] Zafari F, Gkelias A, and Leung K, "A survey of indoor localization systems and technologies," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2568-2599, Apr. 2019. Article (CrossRef Link)

[5] H. Cheng, Y. Zhuang, H. Liu, J. Li, and W. Wang, "A performance evaluation framework for direction finding using BLE AoA/AoD receivers," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3331-3345, Mar. 2021. Article (CrossRef Link)

[6] S. Zhou and J. K. Pollard, "Position measurement using Bluetooth," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp. 555-558, May. 2006. Article (CrossRef Link)

[7] J. P. Van Marter, A. G. Dabak, N. Al-Dhahir, and M. Torlak, "Support Vector Regression for Bluetooth Ranging in Multipath Environments," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11533 – 11546, July. 2023. Article (CrossRef Link)

[8] C. Huang, Y. Zhuang, H. Liu, J. Li, and W. Wang, "A performance evaluation framework for direction finding using BLE AoA/AoD receivers," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3331-3345, Mar. 2021. Article (CrossRef Link)

[9] Q. Lin, J. Son, and H. Shin, "A Mean Optimization Filter to Improve Bluetooth AoA Indoor Positioning Accuracy for Ship Environments," in *Proc. of International Conference on Indoor Positioning and Indoor Navigation (IPIN) -WiP*, no. 1, pp. 1-16, Sep. 2022. Article (CrossRef Link)

[10] Q. Lin, J. Son, and H. Shin, "A self-learning mean optimization filter to improve Bluetooth 5.1 AoA indoor positioning accuracy for ship environments," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 1, pp. 59-73, Mar. 2023. Article (CrossRef Link)

[11] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi, "DNN-based prediction model for spatio-temporal data," in *Proc. of the 24th ACM SIGSPATIAL international conference on advances in geographic information systems*, vol. 2016, no. 92, pp. 1-4, Oct. 2016. Article (CrossRef Link)

[12] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, et al, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, no. 53, pp. 1-74, Mar. 2021. Article (CrossRef Link)

[13] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig, "Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 9, no. 4, pp. 235-245, Mar. 2019. Article (CrossRef Link)

[14] K. E. ArunKumar, D. V. Kalaga, C. M. S. Kumar, M. Kawaji, and T. M. Brenza, "Comparative analysis of Gated Recurrent Units (GRU), long Short-Term memory (LSTM) cells, autoregressive Integrated moving average (ARIMA), seasonal autoregressive Integrated moving average (SARIMA) for forecasting COVID-19 trends," *Alexandria Engineering Journal*, vol. 61, no. 10, pp. 7585-7603, Oct. 2022. Article (CrossRef Link)

[15] M. Sajjad, Z. A. Khan, A. Ullah, T. Hussain, W. Ullah, M. Y. Lee, and S. W. Baik, "A novel CNN-GRU-based hybrid approach for short-term residential load forecasting," *IEEE Access*, vol. 8, no. 1, pp. 143759-143768, July. 2020. Article (CrossRef Link)

[16] E. Yazan and M. F. Talu, "Comparison of the stochastic gradient descent based optimization techniques," in *Proc. of International Artificial Intelligence and Data Processing Symposium (IDAP)*, pp. 1-5, Nov. 2017. Article (CrossRef Link)

[17] F. Zou, L. Shen, Z. Jie, W. Zhang, amd W. Liu, "A sufficient condition for convergences of adam and rmsprop," in *Proc. of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 11127-11135, June. 2019. Article (CrossRef Link)

[18] S. Yiu, M. Dashti, H. Claussen, et al, "Wireless RSSI fingerprinting localization," *Signal Process*, vol. 131, no. 1, pp. 235-244, Feb. 2017. Article (CrossRef Link)

[19] P. Sambu and M. Won, "An experimental study on direction finding of Bluetooth 5.1: Indoor vs outdoor," in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1934-1939, May. 2022. Article (CrossRef Link)

[20] Nordic Semiconductor, "nWP036 -Direction Finding,". [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fnwp_036%2FWP%2Fnwp_036%2Fintro.html

[21] R. Ward, X. Wu, and L. Bottou, "Adagrad stepsizes: Sharp convergence over nonconvex landscapes," *Journal of Machine Learning Research*, vol. 21, no. 219, pp. 1-30, Dec. 2020. Article (CrossRef Link)

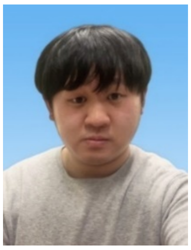
[22] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of local-update SGD algorithms," *Journal of Machine Learning Research*, vol. 22, no. 213, pp. 1-50, Sep. 2021. Article (CrossRef Link)

[23] Y. Shin, Q. Lin, and J. Son, "A Deep Learning Method for IQ Fingerprint Map Based Indoor Positioning using Bluetooth 5.1 Technology in Ship Environments," in *Proc. of APIC-IST*, pp. 257-259, June. 2023.

[24] Adrian's Blog, Scikeras 0.11.0. documentation, [Online], Available: https://adriangb.com/scikeras/stable/, Accessed on: Mach. 12, 2024.

[25] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," in *Proc. of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 374-382, June. 2019. Article (CrossRef Link)

Yootae Shin has completed his undergraduate studies in the Division of Marine Information Technology, Korea Maritime and Ocean University, Busan, South Korea. His research interests include indoor positioning, deep learning, and maritime intelligence.



Qianfeng Lin is currently pursuing Ph.D. with the Department of Computer Engineering, Korea Maritime and Ocean University, Busan, South Korea. His research interests include indoor positioning, deep learning, machine learning, data mining, and maritime intelligence.

Jooyoung Son received the Bachelor of Science degree from the Seoul National University, Seoul, Korea in 1985, the Master of Engineering degree in Computer Engineering from the Seoul National University in 1993, and the Ph.D. degree in Computer Engineering from the Seoul National University in 1997. He was a senior researcher at Media Communication Laboratory, LG electronics during 1985 and 1998. He is currently a professor at Division of Marine IT Engineering, Korea Maritime and Ocean University, Busan, Korea since 1998. His research interests include in-ship localization, telecommunication systems in ships and at sea, and optimization for maritime affairs.