國立臺灣大學電機資訊學院電信工程學研究所
碩士論文

Graduate Institute of Communication Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

利用自走車輔助學習以改善行動網路室內定位效能

Improving Indoor Localization for Cellular Networks with Robot-Assisted Learning

洪健豪

Chien-Hao Hung

指導教授：謝宏昀 博士

Advisor: Hung-Yun Hsieh, Ph.D.

中華民國 111 年 9 月
September 2022

doi:10.6342/NTU202203944

致謝

時間蠻快就過了，我花了五年才完成我的碩士學位，其中經歷了對未來的迷茫，家裡的變故，但也碰到許多的人及事務的幫忙，精神上或物資上的幫忙，其中最感謝的是我的父母在我求學期間的支持，感謝他們在我求學期間的付出及幫助，在我想休學不讀時，持續給我支持，也感謝 謝宏昀老師，在我家裡變故時還想著有什麼方法能夠最大程度的幫助我，感謝老師的著想。感謝我修課時的好夥伴，恩妤、聖皓在修課當個凱瑞的隊友，幫忙應付機器學習相關的課程，感謝維方、泓弦及梓維學長在修課上意見幫助，看著學長們畢業也給了我一些信心相信自己也能完成自己的學位，感謝俊翔在做計劃時的凱瑞，後面我家裡發生變故就扛起計畫的責任並且準時完成學位，真的很厲害。感謝政旻、世紀、承翰、易錡在完成論文上的各種幫助，政旻幫忙邀請共同的口委，世紀跟易錡兩個擔任搞笑擔當，讓我對口試沒那麼害怕，承翰則是給我很大的推動，邊工作邊寫論文還寫得很快，讓我對自己的進度也不敢怠慢。最後感謝我自己，每次想放棄時都沒有真的放棄。

2022/09/23 洪健豪

doi:10.6342/NTU202203944

摘要

對室內定位的需求急劇增加，許多應用都需要高精度的室內定位技術，如智能工廠、智能配送、智能旅遊等。和基於測距的室內定位技術相比，基於學習的室內定位技術更能適應環境限制，基於學習的室內定位定位技術對設備的要較低。然而，基於學習的室內定位通常需要耗費人力和耗費時間來建立指紋數據集。此外，隨著時間、季節和溫度的變化，需要對模型進行實時調整。在這兩種情況下，都需要重建指紋數據集來訓練模型，這增加建立系統的成本。我們應用該機器人構建了一個可以邊走邊採集 LTE 無線電特徵的系統，並利用 SLAM 演算法計算出的軌跡數據來輔助無線電特徵的標註。我們提出了 MICNN+RNN 串接模型，串接模型的性能可以達到 0.879 m，實現了亞米級室內定位。對於 MICNN 模型，我們提出應用基於參數的遷移學習方法將從源域系統學到的知識遷移到目標域，該方法可以將 MICNN 的模型性能提高 6.7%平均距離誤差 (MDE)。分析不同縮放器對模型性能的影響，我們發現 MinMax 縮放器有助於目標模型性能和微調模型性能。

doi:10.6342/NTU202203944

ABSTRACT

The demand for indoor positioning has increased dramatically. Many applications require high-precision indoor localization technology, such as smart factories, smart deliveries, smart tours, etc. Compared with ranged-based indoor localization technologies, learning-based indoor localization technologies are more adaptable to environmental constraints, and learning-based indoor positioning technologies have lower requirements for instruments. However, learning-based indoor localization is often labor-intensive and time-consuming to build a fingerprint dataset. Also, with the change in time, season and temperature, the model needs to be adjusted in real-time. In both cases, the fingerprint dataset needs to be rebuilt to train the model, which increases the cost of building the system. We applied the robot to build a system that can collect LTE radio features while walking, and use the trajectory data calculated by the SLAM algorithm to assist in radio feature annotation. We propose the MICNN+RNN cascaded model, and the performance of the cascaded model can reach 0.879 m, which achieves sub-meter indoor localization. For the MICNN model, we propose to apply the parameter-based transfer learning method to transfer the knowledge learned from the source domain system to the target domain, and this method can improve the model performance of MICNN by 6.7% for mean distance error (MDE). Analyzing the effect of different scalers on model performance, we found that the MinMax scaler is helpful for the target model performance and fine-tuned model performance.

doi:10.6342/NTU202203944

TABLE OF CONTENTS

doi:10.6342/NTU202203944

doi:10.6342/NTU202203944

doi:10.6342/NTU202203944

LIST OF TABLES

doi:10.6342/NTU202203944

LIST OF FIGURES

doi:10.6342/NTU202203944

doi:10.6342/NTU202203944

doi:10.6342/NTU202203944

CHAPTER 1

INTRODUCTION

With the development of robotics, more and more robotics applications are being developed, such as smart factories, disaster relief, robot delivery, smart warehouses, etc. These applications are built on a high-accuracy localization system, and these applications need to track the exact location of the robots. According to industry 4.0 requirements, accurate arrival at the exact location to avoid wasting time is an important part of smart factory operation. Therefore, sub-meter indoor localization has become the standard for a localization system. Localization can be divided into outdoor positioning and indoor positioning. For outdoor localization, GNSS and GPS systems are a good solution to meet the demand for high-accuracy localization. Indoor localization does not have a standard approach or system; GNSS and GPS signals are unavailable for the indoor environment, and the indoor environment is more complex and prone to signal interference. Therefore, sub-meter indoor localization is still full of challenges.

There are many papers discussing radio-based indoor localization technologies, including Wi-Fi, Ultra Wideband (UWB), Zigbee, Bluetooth, mobile network, etc. Although these devices can achieve indoor positioning, not all of them provide high accuracy. Wi-Fi, Zigbee, and Bluetooth use an open bandwidth band, and signal interference in this bandwidth is usually the greatest. In addition, the coverage area must also be taken into consideration, as the transmission distance of UWB is small (< 30 meters), which makes the UWB localization system expensive. Zigbee and Bluetooth use small bandwidth and transmission power, and more interference in the frequency band is not suitable for localization. In summary, we consider the mobile network as our radio localization platform. With the development of 5G, the mobile network has a great improvement, such as multiple antennas, large bandwidth, beamforming technology, etc. The mobile network has great potential to provide a high precise localization accuracy.

According to different algorithms, the localization system can be divided into ranging-based and learning-based approaches. The Ranging-based localization approach includes angle of arrival (AoA), time difference of arrival (TDoA), and time of arrival (ToA), all of which require several base stations to work together, and the system are easily affected by the environment. In additional, the ranging-based system can not be applied in a non-line-of-sight (NLoS) environment. With

doi:10.6342/NTU202203944

the rapid development of artificial intelligence, the learning-based localization approach has become a new solution. The learning-based localization system does not require several base stations for localization. In addition, the learning-based localization system is resistant to noise interference. However, there are some challenges for the learning-based localization system.

For a learning-based localization system, localization deployment is an important part, which includes building fingerprint datasets, model training, etc. The fingerprint collection method can be broadly divided into a fixed-point collection or walking collection. Fixed-point fingerprint collection can facilitate fingerprint labeling. However, the fixed-point collection method makes the fingerprints collected at the same location highly correlated, which makes the model overfitting easily. Therefore, the walking collection method becomes our choice, but this method makes labeling fingerprints difficult. Therefore, we combine the SLAM algorithm to assist in fingerprint labeling. There is another problem for fingerprint collection. Fingerprint collection is a labor-intensive and time-consuming task. The localization system can generate 20 fingerprints every second, and it takes about 10 minutes to collect a trajectory dataset. The collection task includes the setting of turtlebot3 and the setting of the computer. After walking around, we had to spend 20 minutes charging the turtlebot3. In our experiment, it took about 4 hours to collect 8 trajectory datasets. Collecting fingerprints is a very time-consuming task.

To save the cost of collecting fingerprint datasets, we propose a transfer learning approach that uses an existing localization system to improve the performance of the target domain system. The contributions of this thesis are as follows:

1.) We have built a system that combines robots with LTE base stations. This system is able to collect LTE radio features while moving. The coordinates obtained by SLAM algorithm are used to assist in labeling the fingerprints.

2.) We apply the transfer learning method to improve the localization performance of the target domain.

3.) For the localization system, we use LTE subcarrier amplitudes and subcarrier phase differences as input features and propose to use the MICNN-RNN model to optimize the localization system.

4.) Analyze the effect of feature distributions on the fine-tuning model and target model.

The rest of this thesis is as follows: Chapter 2 introduces the localization system, related works, robot setup, LTE setup, and machine learning related work. Chapter 3 explores the accuracy of different localization models and introduces our experimental approach and proposed our localization system. The main focus

doi:10.6342/NTU202203944

of this thesis is how to get better indoor localization performance with a little site survey, which will be described in Chapter 4. We try to apply data augmentation and transfer learning to achieve this goal in Chapter 4. We also analyze the effect of feature distribution on the performance of the model in Chapter 4. In Chapter 5, we evaluate the approach which will described in Chapter 4 with the different amount of domain trajectory datasets. Finally, we summarize our works in this thesis and suggest some possible approaches in 6.

doi:10.6342/NTU202203944

CHAPTER 2

RELATED WORK

2.1 System Introduction

Figure 1 shows a typical indoor scenario. Generally, the environment can be divided into several cells to form the cell set $S_c = c_i | i = 1, 2, .., l$, where l is the number of cells.c_j center position is defined as $p_j = (x_j, y_j)$. In addition, we want to locate the object position on the map at time t as $p_t = (x_t, y_t)$ and define the instantaneous motion data of the object at time t as u_t.

In the case of indoor tracking, the motion data of the object is usually able to improve the accuracy significantly. In our scenario, the object is equipped with a soft defined radio capable of receiving the LTE signal and extracting the channel information state from the reference signal of LTE as the feature. We consider the cell-specific reference signals (CRS) as the reference signals to extract the vectors ν_t, which are described in detail in Section 3.1.1.

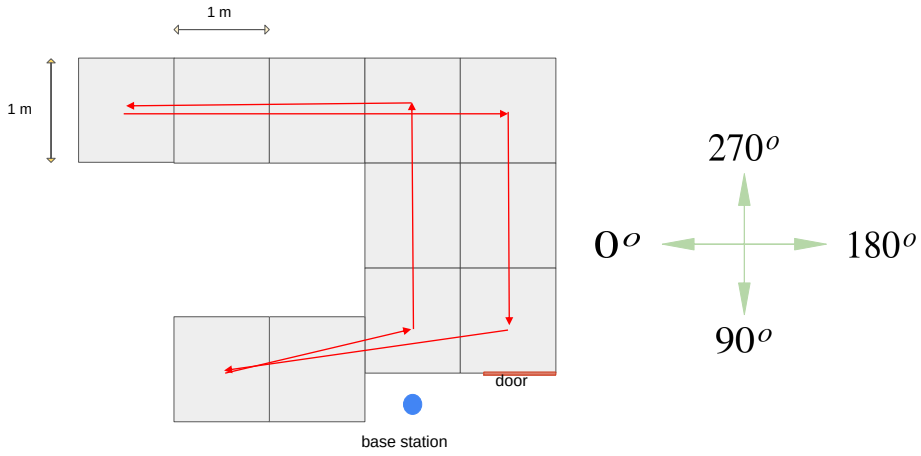


Figure 1: Scenario

2.1.1 Problem Formulation

Figure 2 shows a hidden Markov model (HMM), where x_t represents the hidden state of the tracker, z_t represents the observed state at time t, and u_t is the control signal at time t. In this thesis, x_t represents the object position P_t, and z_t represents the observed CSI vector $H(P_t)$ when the hidden state is x_t, and u_t is our

doi:10.6342/NTU202203944

measured motion data v_t. From the observation of Figure 2, we are available to derive the formula to simulate the real tracking problem, the formula is as follows,

$$x_t = g(x_{t-1}, u_t) + n_t, \tag{2.1}$$

$$z_t = h(x_t) + m_t, \tag{2.2}$$

where $g()$ represents the process function, and $h()$ represents the measurement function, and n_t and m_t represent the process noise and measurement noise respectively.

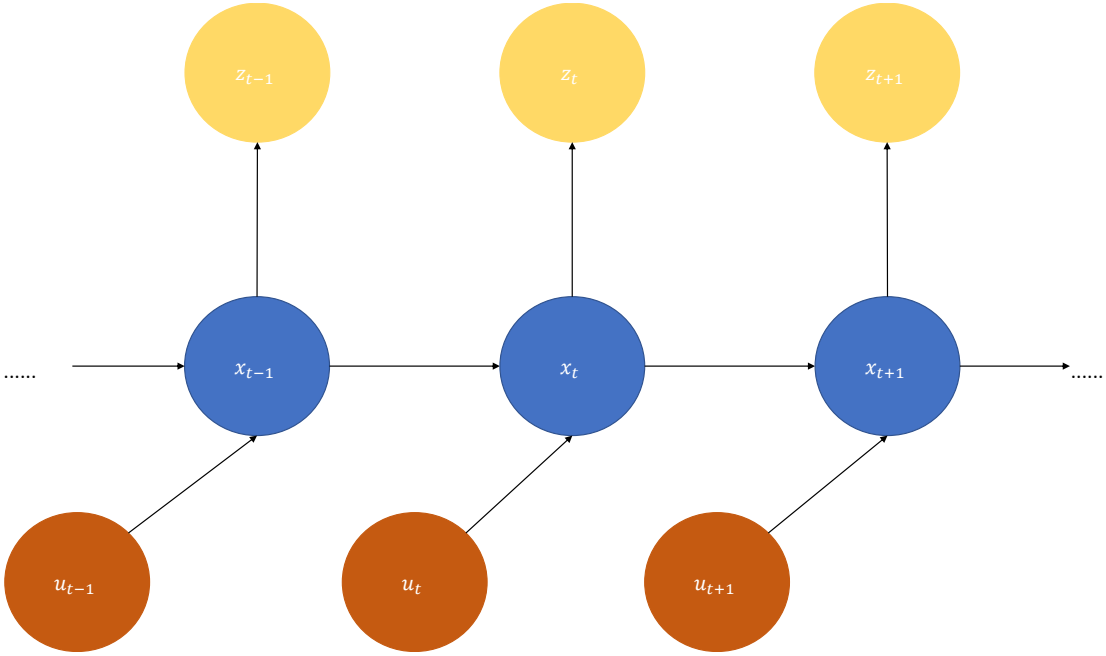


Figure 2: Hidden Markov model

Following the Equation (2.2), considering the case of the time point t, we expect to find a transition function $\hat{h}()$ that leads to the following equation,

$$\hat{P}_t = \int \hat{h}(z_t) * p(z_t) dz_t, \tag{2.3}$$

where $p(z_t)$ represents the probability density function, and the expected value of that equation is the estimated position. It is also desired that the derived $\hat{P}_t$ achieves the following objective function,

$$\begin{aligned} \min_{\hat{h}()} \quad & \frac{1}{M} \sum_{t=1}^{M} ||P_t - \hat{P}_t|| \\ \text{s.t.} \quad & Eq.2.3 \\ & P_t, \hat{P}_t \in \mathbf{A}, \forall t \end{aligned} \tag{2.4}$$

where $\mathbf{A}$ denotes the feasible tracking area.

doi:10.6342/NTU202203944

The above discussed how to use the observed state at a single time point to estimate the position. The following discusses the help of obtaining the continuous-time series observation states and the motion data for tracking. The objective of tracking is able to obtain a belief probability distribution when we use the existing observation states and measurement data, as follows,

$$f(x_t|z_{1:t}, u_{1:t}),$$

where $u_{1:t}$ and $z_{1:t}$ represent motion data sequence and observed state sequence, respectively. In general, the traditional method use particle filter and Kalman filter to estimate the hidden state, in the case of the great progress of artificial intelligence, we try to find a function $\hat{g}()$ as follows,

$$\hat{x}_t = \hat{g}(\{\hat{h}(z_i), i = 1, 2, 3, ...t\}), \tag{2.5}$$

where $\{\hat{h}(z_i), i = 1, 2, 3, ...t\}$ represents the sequence $z_{1:t}$ individuals after the transformation function $\hat{h}()$. By using this transformation function $\hat{g}()$, several data can be combined to form the belief probability distribution, and the expected value of that, P'_t, can be more accurate than $\hat{P}_t$ estimated by a single observation, and the following objective function,

$$\begin{aligned} \min_{\hat{g}()} \quad & \frac{1}{M}\sum_{t=1}^{M} ||P_t - \hat{P}'_t|| \\ \text{s.t.} \quad & Eq.(2.5), \\ & P_t, P'_t \in \mathbf{A}, \forall t. \end{aligned} \tag{2.6}$$

According to the objective Equation (2.4) and Equation (2.6), it is difficult to find $\hat{h}()$ and $\hat{g}()$ by using traditional mathematical models. Due to the development of artificial intelligence, it is possible to use neural network models to describe $\hat{h}()$ and $\hat{g}()$.

To sum up, we divide the tracking problem into two problems, one is to estimate the position from the features at a single time point, and the other is to use time-series data to improve the tracking accuracy. In the next chapter, we introduce various models for the tracking problem and propose models based on the features.

2.2 Robot Setup

2.2.1 Robot Operating System (ROS)

With the development of robot technology and scale, the programming software for robots is a difficult task. Different robots often have different hardware requirements, which makes it difficult to reuse software. Traditionally, managing a robot requires managing drivers, protocols, abstraction management, and

doi:10.6342/NTU202203944

more. The Robot Operating System was developed to make robot development and management easier. The ROS [1] is not an operating system, but a software platform that can communicate between the operating system and the code. ROS also provides some tools and libraries for data transmission. We use the SLAM library which is provided by ROS to obtain maps, moving track, and LTE CSI.

2.2.2 Simultaneous Localization and Mapping

Simultaneous Localisation and Mapping (SLAM) use data from sensors to calculate its position and trajectory in an unknown environment. SLAM algorithms can continuously construct and update spatial information in two or three dimensions. Turtlebot3 offers two SLAM algorithms, GMapping [2] and Cartographer [3]. We mainly use Cartographer in our application. Cartographer is a graph-based SLAM algorithm that can support 2D or 3D maps and supports multiple sensors. Cartographer is a subgraph-based construction method, so it is effective to reduce environmental disturbances. We use Cartographer to calculate the relative movement coordinates of objects which are manually corrected to become our data labels.

2.2.3 Turtlebot3

Turtlebot3 [4] is a small, affordable, programmable, ROS-based mobile robot for education, research. The goal of Turtlebot3 is to significantly reduce the size and price of the platform without compromising its functionality and quality while providing scalability. Turtlebot3 is capable of running SLAM in its entirety and the SLAM algorithm helps us to collect data for indoor tracking applications.

2.3 LTE Setup

2.3.1 Software Defined Radio

Traditional radio systems can transmit and receive signals through components such as amplifiers, detectors, modulators, demodulators, filters, and mixers. Software Definition Radio [5], as the name implies, uses software settings to transmit and receive data. Compared to commercial base stations, SDRs allow us to dynamically configure different protocols or transmission settings.

2.3.2 OpenAirInterface

Openairinterface (OAI) is a fully open-source experimental platform created by EURECOM to enable the innovation and development of mobile communication networks. The functionalities of the transceiver include base station, access point, mobile terminal, and core network. These functionalities are mainly performed

doi:10.6342/NTU202203944

by the computer through software. OAI provides a complete implementation of all elements of LTE, including user equipment (UE), eNodeB (eNB), and core network. The software implementation of OAI is fully compliant with 3GPP LTE protocol standards. OAI can build emulated physical layer links and use them on the same computer, and OAI can also build actual radio channels via SDR.

2.4 Related Work about Machine Learning

2.4.1 Recurrent neural network

In many applications such as voice, text, video, etc., use sequential features as input of the ML model. Compared to DNN and CNN which use a single text or image as input features, recurrent neural network(RNN) uses sequential features to obtain better performance.

For vanilla RNN, given a sequence $[x_1, x_2, ..., x_n]$ as model input, a sequence output $[y_1, y_2, ..., y_n]$ can be derived iteratively from the function, as in the following equation,

$$h_t = \sigma(W_h x_t + U_h h_t + b_h), \tag{2.7}$$

$$y_t = \sigma(U_y h_t + b_y), \tag{2.8}$$

where W_h, U_h, b_h, U_y are the weights and bias learned from the dataset, h_t denotes the hidden state at time point t, σ is activate function. From Figure 3, we can see that the output is not only related to the input at the current time point, but the inputs of the previous time also affect the output of the current time point.

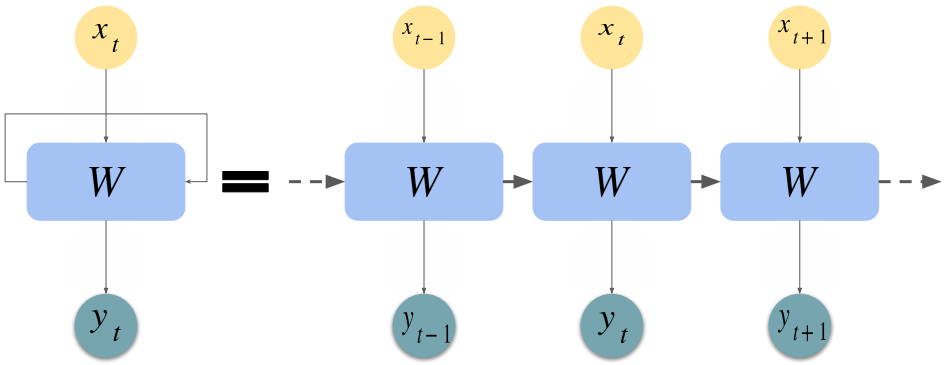


Figure 3: Naive recurrent neural network

doi:10.6342/NTU202203944

2.4.1.1 Long Short-Term Memory

Since there is a gradient vanishing problem about vanilla RNN, it is difficult to save the information in the inputs at earlier time points, and the hidden state of vanilla RNN usually stores the data close to the adjacent output time points. Compared with vanilla RNN, Long short-term memory (LSTM) solves the problem that RNNs cannot learn long-term memory. The LSTM Cell is as follows Figure 4, one LSTM cell contains two states, three gates.

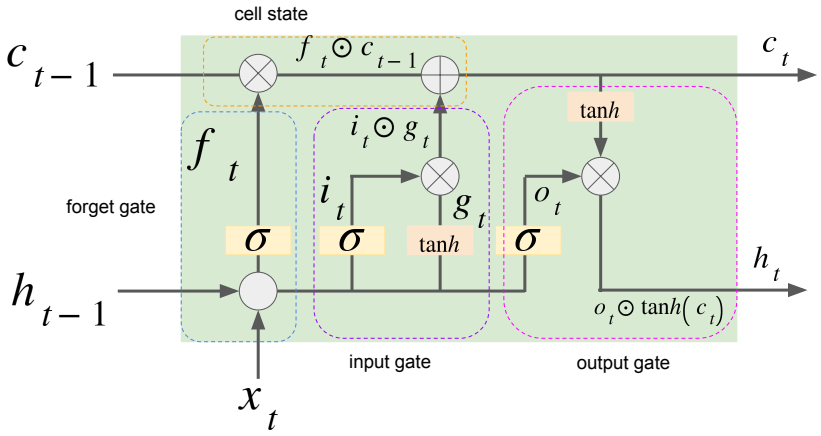


Figure 4: Long short-term memory cell

The function at each step is defined as follows:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
g_t &= tanh(W_g x_t + U_g h_{t-1} + b_g), \\
c_t &= f_t \odot c_{i-1} + i_t \odot g_t, \\
h_t &= o_t \odot tanh(c_t),
\end{aligned}
$$

where σ and $tanh$ represent the sigmoid function and hyperbolic tangent function, W,U and b are trained matrix and bias, h_t and c_t denote the hidden state and cell state. Compared with vanilla RNN, LSTM has an additional long-term memory c_t to solve the problem of gradient vanishing.

2.4.1.2 Gated Recurrent Unit

Although LSTM solves the vanilla RNN, the slow execution speed is also the drawback of LSTM. The concept of Gated Recurrent Unit (GRU) was proposed

doi:10.6342/NTU202203944

in [6], where the forget gate and input gate were combined into the update gate, and the cell state and hidden state were combined. Therefore, the weight number of GRU is less than LSTM to reduce execution speed. The accuracy of GRU is better than LSTM when trained with specific datasets.

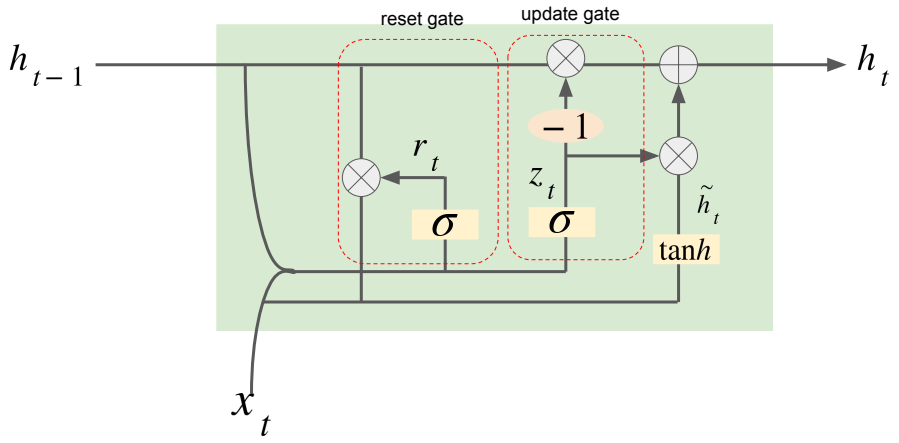


Figure 5: Gated Recurrent Unit

The calculation process is as follows:

$$
\begin{aligned}
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r), \\
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z), \\
\tilde{h_t} &= tanh(W x_t + r_t U h_t + b_h), \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t}.
\end{aligned}
$$

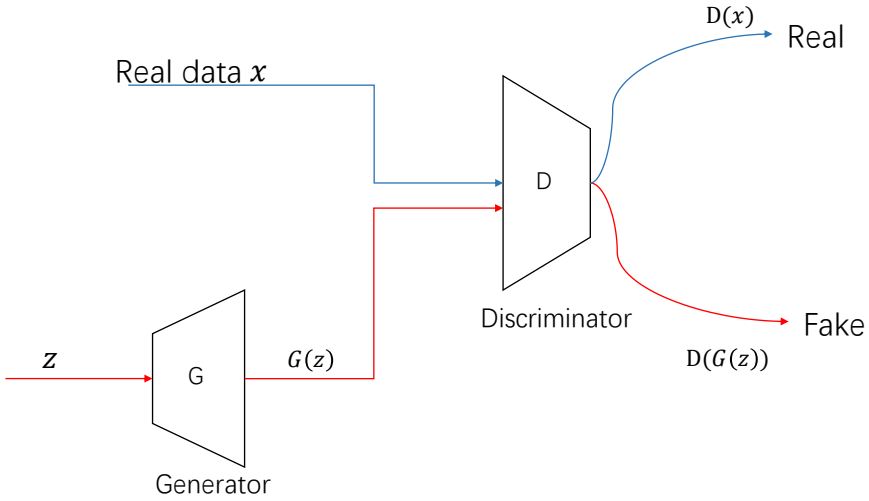
Both GRU and LSTM use the concept of gates, while LSTM uses forget gate and input gate to control the deletion of memory and update of input, GRU only uses update gate to do so. The LSTM output data is generated by the tanh function and the output gate, and the GRU uses the update gate to control the output.

2.4.2 Generative Adversarial Network

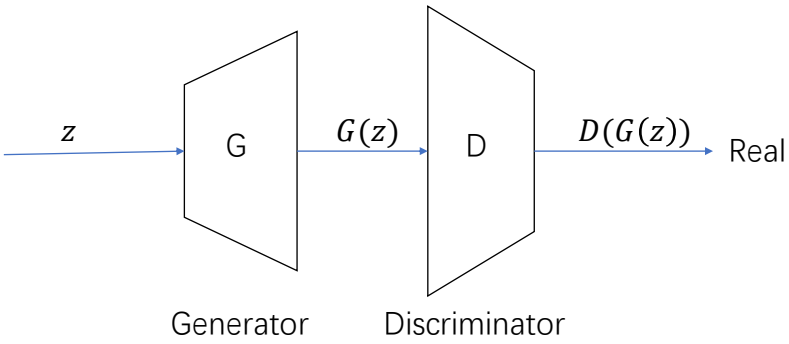
Generative adversarial networks (GAN) were proposed by Goodfellow et al. in 2014.GAN achieves unsupervised learning through a set of model adversaries.GAN is mainly composed of two models, discriminator, and generator. The learning goal of the discriminator is to detect whether the data is real or not, while the learning goal of the generator is to generate data that can fool the discriminator. During the adversarial process, the two models improve their capabilities together.

Figure 6 shows the training process of generator and discriminator. From Figure 6(a), we can see that the training process of the discriminator is based on

doi:10.6342/NTU202203944



(a) Training discriminator process of GAN



(b) Training generator process of GAN

Figure 6: Training process of GAN

doi:10.6342/NTU202203944

two kinds of data, fake data and real data. Therefore, the loss function for training the discriminator can be written as the following equation,

$$\max_D E_{x\sim q(x)}[logD(x)] + E_{z\sim p(z)}[log(1 - D(G(z)))], \tag{2.9}$$

where $E_{x\sim q(x)}[logD(x)]$ represents the binary cross entropy of the real data and $E_{z\sim p(z)}[log(1-D(G(z)))]$ represents the binary cross entropy of the fake data. Gradient descent algorithm to find the minimum value in deep learning applications, so the function is displayed as follows,

$$\min_D -E_{x\sim q(x)}[logD(x)] - E_{z\sim p(z)}[log(1 - D(G(z)))]. \tag{2.10}$$

Observe that the loss function of generator from Figure 6(b) is represented as follows,

$$\max_G E_{z\sim p(z)}[log(D(G(z)))]. \tag{2.11}$$

To optimize the function using gradient descent algorithm, the loss function is written as the following equation,

$$\min_G -E_{z\sim p(z)}[log(D(G(z)))]. \tag{2.12}$$

Algorithm 1 Minibatch stochastic gradient descent training of GAN

Require: 1.) The number of steps to apply to the discriminator, k, is a hyperparameter. $k = 1$ is the least expensive option. **Output** 1) θ_D is the parameters of discriminator D ; 2) θ_G is the parpameters of generator G

1: **for** t iterations **do**
2: **for** k steps **do**
3: Sample minibatch of m noise sample $\{z^1, z^2, ..., z^m\}$ from noise prior $p_g(z)$.
4: Sample minibatch of m example $\{x^1, x^2, ..., x^m\}$ from data generating distribution $p_{data}(x)$.
5: Update the discriminator by descending its stochastic gradient:

$$\nabla_{\theta_d} \frac{-1}{m} \sum_i^m [logD(x^i) + log(1 - D(G(z^i)))].$$

6: **end for**
7: Sample minibatch of m noise sample $\{z^1, z^2, ..., z^m\}$ from noise prior $p_g(z)$.
8: Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_i^m [log(1 - D(G(z^i)))].$$

9: **end for**

doi:10.6342/NTU202203944

2.4.3 Transfer Learning

Transfer learning is used extensively in many research. In the image recognition field, the convolutional layers in a pre-trained CNN model are often transferred to a new model as parameters initialization. In natural language research, where various semantics are highly correlated, the model responsible for semantic discrimination is often transferred to the new model as parameters initialization. The transfer learning approach enables applications with a small amount of data to achieve good recognition results, such as few-shot learning [7], and the convolutional layers in a model usually require a large number of models to be trained to achieve better results, and the use of pre-trained convolutional layers can overcome the problem of insufficient training data.

According to the learning solution, it is proposed in [8] that transfer learning can be divided into four approaches, feature-based transfer learning, parameter-based transfer learning, instance-based transfer learning, and relational-based transfer learning. This is called instance-based transfer learning, and trAdaboost [9] is a well-known method to weight the training source data that is similar to the target domain data. Domain adaptation [10] is a common approach in feature-based transfer learning, and the source domain data and target domain data are projected into an approximate feature space to help predict the data in the target domain. Sharing parameters [11], [12] in the model and transferring parameters from the source domain model to the target model for fine-tuning [13] is a common parameter-based transfer learning.

2.5 Related Work

2.5.1 Indoor Localization

There are many different studies in the indoor localization field. There are two types of positioning methods, the ranging-based method, and the learning-based method. In this thesis, we mainly focus is on the learning-based indoor localization method.

2.5.1.1 Localization System

Table 1 shows the related works on the indoor localization system. The features used for indoor localization are applied according to different wireless technologies. The received signal strength indicator (RSSI) is a common used feature, which is common in Bluetooth, Wi-Fi, and Zigbee, as mentioned in [14], [15]. RSSI is quantified data, and the localization system using RSSI as features usually requires more base stations to send signals to obtain a better localization accuracy. Channel

doi:10.6342/NTU202203944

System	Architecture	Technique	Feature	Accuracy
SLN + FN [16]	DNN	LTE	CSI amplitued	0.47 m
SDR-Fi [17]	1DCNN	Wi-Fi	CSI amplitude	0.99 m
MT Hoang [14]	LSTM	Wi-FI	RSSI	4m
X. Peng [18]	WKNN	Wi-Fi	CSI amplitude	2.18 m
PhaseFi [20]	RBF	Wi-Fi	CSI phase	1.08 m
G. Pecoraro [21]	WKNN	Wi-Fi	CSI+RSRP	0.14 m
Y. Zhang [19]	LSTM	Wi-Fi	CSI amplitude + CSI calibration phase	1.03 m
DL-RNN [15]	LSTM	Wi-Fi	RSSI	3.0556 m
Proposed method	1DCNN + LSTM	LTE	CSI amplitude + CSI phase difference	0.879 m

Table 1: Environmental adaptation

state information is the channel response, which reflects the energy changes and phase changes between reception and transmission. The real part of CSI is used as a feature mentioned in [16], [17], [18], [16]. The phase part of CSI is mixed with some noise, but it still provides some information that requires additional processing, as mentioned in [19]. In LTE systems, the real part of CSI represents the amplitudes of the subcarriers and the imaginary part of CSI represents the phases of the subcarriers. We use the amplitude and phase difference of the subcarriers as inputs to our positioning model. In addition, the model can obtain the better localization accuracy, so there are some similar approaches mentioned in [16], [14], [19], [15]. We use trajectory datasets as training data and test data, and the data is continuous in time, so we use LSTM to fuse the data in the time domain.

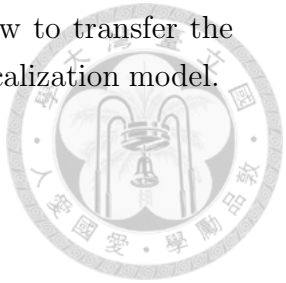
2.5.1.2 Environmental Adaptation

Environmental adaptation is a topic of indoor localization. Environmental adaptation can be divided into two main situations, adaptation in the same field [22] and system adaptation in a different field [23]. The localization system in a site is not always applied to the same site. Temperature, humidity, and environmental changes can affect the features received. Therefore, the learning-based indoor localization system needs to be adjusted at intervals. The main purpose of adapting to the environment in different fields is to reduce the cost of system deployment. Fingerprint collection for indoor localization is a labor-intensive and time-consuming task. Some papers propose to use trajectory data for labeling data [24], [25]. The use of a transfer learning approach is a novel approach to deploy a localization system with a complete site survey to a new field.

In [23], the authors propose a transfer learning method based on clustering algorithms (e.g., K Nearest Neighbor algorithm) that can effectively transfer a localization system with the complete site survey to the target domain with a small amount of training data. However, the method proposed in [23] is still

doi:10.6342/NTU202203944

somewhat limiting for the deep learning model. We explore how to transfer the model based on the deep learning model into a target domain localization model.



doi:10.6342/NTU202203944

CHAPTER 3

LEARNING-BASED INDOOR LOCALIZATION

3.1 Feature

3.1.1 LTE Subcarrier Amplitude

In this thesis, we consider the use of a frequency division duplexing (FDD) system. The base station is equipped with a Mac Pro which connected to a software-defined terminal and implements OpenAirInterface (OAI), while the UE is a turtlebot3 with an Up board and USRP B210. Also it implements OAI. The UE obtains the LTE subcarrier channel response from the CRS which is transmitted by the base station. It is referred to as the LTE CSI.

For tracking methods that require fingerprints on offline phase, the UE is usually used to collect fingerprints at the center of each cell, which has the disadvantage of requiring a large cost to build the fingerprint dataset. Another mode of fingerprint collection is also widely discussed it is about algorithms or instruments to obtain a higher accuracy of the object's path trajectory and receiving reference signals while the object is moving. In this thesis, we use the SLAM algorithm of ROS to obtain accurate map information and transform it as the ground truth of the training data, and label the data as the cell where it is located. Figure 7 shows a resource block in a typical LTE subframe, the interval space between each adjacent subcarrier is 15 kHz, a time slot contains seven OFDM symbols, a symbol can span 12 consecutive subcarriers, where each base station will be fixed time to send cell-specific reference signal(CRS), this signal is used to measure CSI.

Given collection location P_t and time slot t, we can obtain CSI from the n^{th} transmit antenna at the i^th subcarrier, is given by

$$h_i^n(P_t) = |h_i^n(P_t)|e^{j\measuredangle h_i^n(P_t)},$$

where $|h_i^n(P_t)|$ and $\measuredangle h_i^n(P_t)$ represent amplitude and phase, respectively. The phase part is more likely to be disturbed by the angle of incidence, diffuse radiation, etc. In this thesis, the component vectors of the CSI real numbers are used as

doi:10.6342/NTU202203944

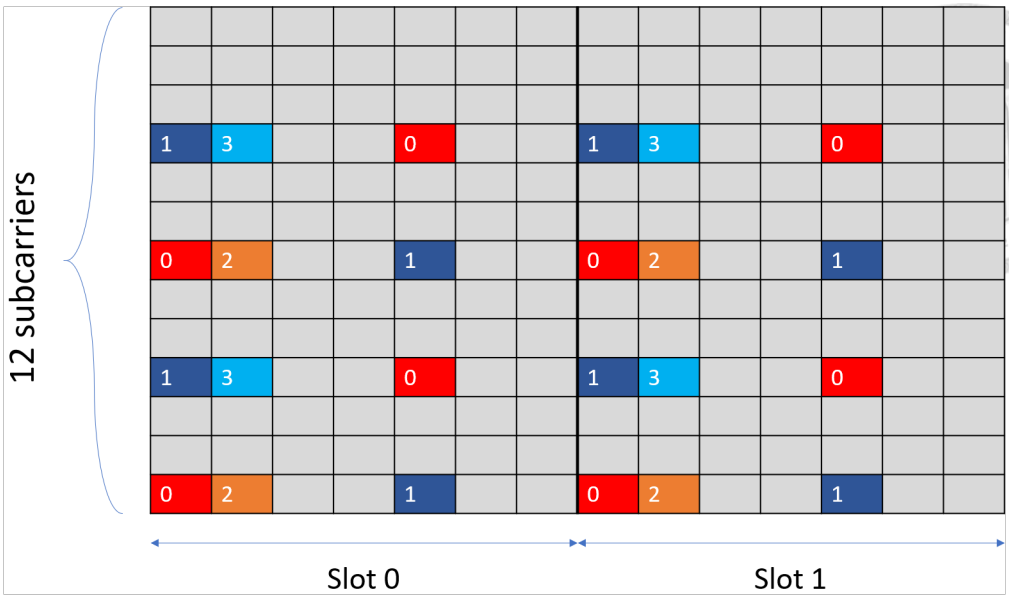


Figure 7: LTE CRS whitin a subframe

follows,

$$H(P_t) = \begin{bmatrix} |h^1_{crs_1}(P_t)| & |h^1_{crs_2}(P_t)| & \dots & |h^1_{N_c}(P_t)| \\ |h^2_{crs_1}(P_t)| & |h^2_{crs_2}(P_t)| & \dots & |h^2_{N_c}(P_t)| \\ \vdots & \dots & \dots & \vdots \\ |h^{N_t}_{crs_1}(P_t)| & |h^{N_t}_{crs_2}(P_t)| & \dots & |h^{N_t}_{N_c}(P_t)| \end{bmatrix},$$

where N_t and N_c denote total number of transmit antennas and CRSs, CRS_k represent the k^{th} CRS, P_t denotes the location where the fingerprint are collected. The $H(P_t)$ vector depends on the number of transmission and reception ports and the number of resource blocks according to the transmission bandwidth. In addition, we also do not consider the imaginary part as a feature, because the phase is more susceptible to noise interference and random jitters. In the case of a 10 MHz transmission bandwidth, 50 resource blocks are used, given one antenna port, $N = 200(25RBs \times 2CRS \times 2positions \times 2receiveport)$ CRSs are used to estimate the complex channel gain.

Ideally, the observed value $H(P_t)$ mainly depends on the location [16], and some other factors have a relatively small influence, such as the orientation of the object facing [26], environment changes, etc.

3.1.2 LTE Phase Difference

In most of the positioning systems use CSI as model input feature. The real part of CSI is used as the model input. The real part of CSI represents the energy of the received subcarriers, but the real part of CSI is susceptible to environment,

doi:10.6342/NTU202203944

temperature, humidity, antenna gain, etc. When the real part of CSI is used as a feature vector, the elements in the vector are more correlated with each other. Although using the real part of CSI as model input can achieve good performance, it is still limited. Therefore, we started to find ways to extract more features from CSI as model inputs.

First, we can write the channel response as the following equation,

$$H(f) = \sum_{i=1}^{n} |h_i(f)| e^{j\angle h_i(f)}, \tag{3.1}$$

where $h_i(f)$ represents the channel response of the i^{th} subcarrier. We obtain $CSI_r = [|h_1(f)|, |h_2(f)|, ..., |h_n(f)|]$ and $CSI_i = [\angle h_1(f), \angle h_2(f), ..., \angle h_n(f)]$ vectors according to Equation (3.1).We have already discussed the real part vector CSI_r in Section 3.1.1, so we focus on the imaginary part vector CSI_i here. We write the phase part more carefully. It can be written as the following equation, which is given by [27], [28], [29],

$$\angle h_i(f) = \angle h_i(f) + (\lambda_s + \lambda_p) m_i + \lambda_c + \beta + Z, \tag{3.2}$$

where $\angle h_i(f)$ represents the true phase value of the i^{th} subcarrier, and λ_s, λ_p, λ_c are the phase shift due to sampling frequency offset, the symbol offset, and carrier frequency offset, m_i is the subcarrier index of subcarrier i, β is the initial phase offset due to phase-locked loop, Z is the AWGN with variance σ^2. The phase errors can be written in the following form,

$$\lambda_p = 2\pi \frac{\Delta t}{N}, \lambda_s = 2\pi (\frac{T' - T}{T}) \frac{T_s}{T_\mu} n, \lambda_c = 2\pi \Delta f T_s n, \tag{3.3}$$

where Δt is the symbol boundary delay, N is the FFT size, T' and T are the sampling periods at the receiver and the transmitter, T_μ is the data symbol length, T_s is the symbol length with the guard interval, Δf is the carrier frequency difference between the transmitter and receiver, n is the time offset.

We obtain the phase difference of the adjacent subcarriers which can be written as the following equation,

$$\Delta\angle h_{j-i}(f) = \Delta\angle h_{j-i}(f) + (\lambda_s + \lambda_p) \Delta m_{j-i} + Z, \tag{3.4}$$

where Δm_{j-i} is the index difference of the adjacent subcarriers in cell-specific signals. We know from Figure 7 that Δm is a fixed value. We can also write the true phase difference $\Delta\angle h_{j-i}(f)$ as the following equation,

$$\Delta\angle h_{j-i}(f) = 2\pi (f_j - f_i)\tau, \tag{3.5}$$

where τ is a propagation delay, f_j and f_i are two adjacent subcarriers. In our system, we use CRS as the reference signal, so the adjacent subcarriers interval

doi:10.6342/NTU202203944

is a fixed value 75kHz if transimission mode is one. However,in our system, even if the two furthest points are only within 10 meters of each other, the phase difference will not vary too much. We decided to use the phase difference vector as an input to the model. Finally, we can obtain a additional vector $CSI_{dp} = [\Delta\measuredangle h_1(f), \Delta\measuredangle h_2(f), ..., \Delta\measuredangle h_{N-1}(f)]$ as model input feature.

3.1.3 Feature Analysis

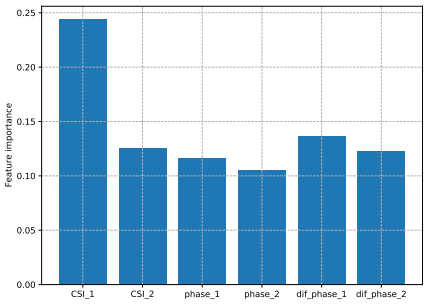
In this thesis, LTE 200 subcarrier amplitudes are used as model inputs. The 199 phase difference values are obtained by subtracting the 200 subcarrier phase values adjacent to each other. We use the Mutual information method [30], [31] to calculate the importance of features for labels. We classify the features extracted from LTE CSI raw data into six categories. The first 100 elements of the CSI raw data are those that have undergone antenna gain. Therefore, we divide LTE CSI subcarrier amplitudes into two categories, the first 100 subcarrier amplitudes, and the last 100 subcarrier amplitudes. We also take phases from the LTE CSI raw data for comparison and divide them into the phases of the first 100 subcarriers and the phases of the last 100 subcarriers. There are a lot of uncertainties in the phase data, which are described in detail in Section 3.1.2. We extract the phase difference from the phases for comparison. The importance of these six on the positioning accuracy is shown in Figure 8.

From Figure 8, the best features for predicting the position are the real part of the first 100 subcarriers, which have undergone antenna gain and can show more subtle channel variations. The last 100 subcarriers have no antenna gain, but the real part of the last 100 subcarriers still give some slight improvement in our experiments. Observe from the Figure 8 that the phase of the subcarriers should also provide some useful information, but the phase of the subcarriers is actually more uncertain. Figure 9 shows a phase difference and the phase distribution. The phase difference is less uncertain than the phase, so we expect the phase difference to provide more information and reduce confusion for the model. From Figures 8, it is observed that the phase difference can provide information about the distance between the observation point and the base station,and the random offsets are subtracted from phase.

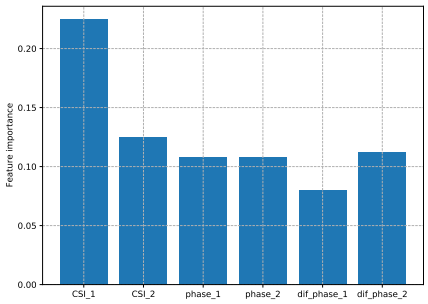
3.1.4 Feature Normalization

In this section, we discuss the input data we use, the feature scaling is also an important part of training the model. Using the data without scaling will cause the contour of the loss function to be elliptical. The gradient direction is perpendicular to the contour line, which may cause the gradient to fall in a direction other than

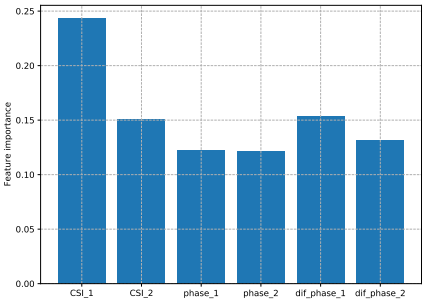
doi:10.6342/NTU202203944



(a) Feature selection for label



(b) Feature selection for position x-axis



(c) Feature selection for position y-axis

Figure 8: Using mutual information to calculate the importance for features

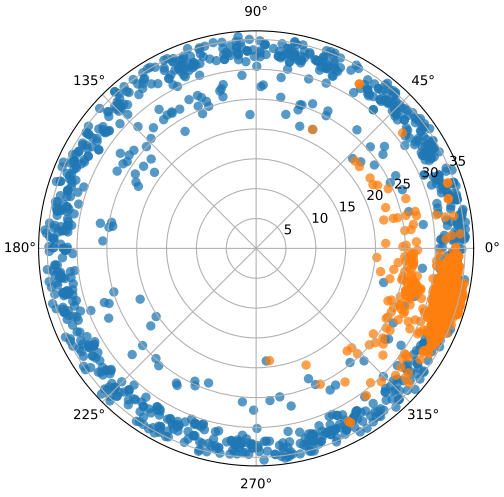


Figure 9: The domain 3 phase distribution and the phase difference distribution

doi:10.6342/NTU202203944

the local minimum, as shown in Figure 10. Training the model without feature scaling will cause the model to take more time to converge or even fail to converge. The common feature scaling is normalization and standardization respectively.

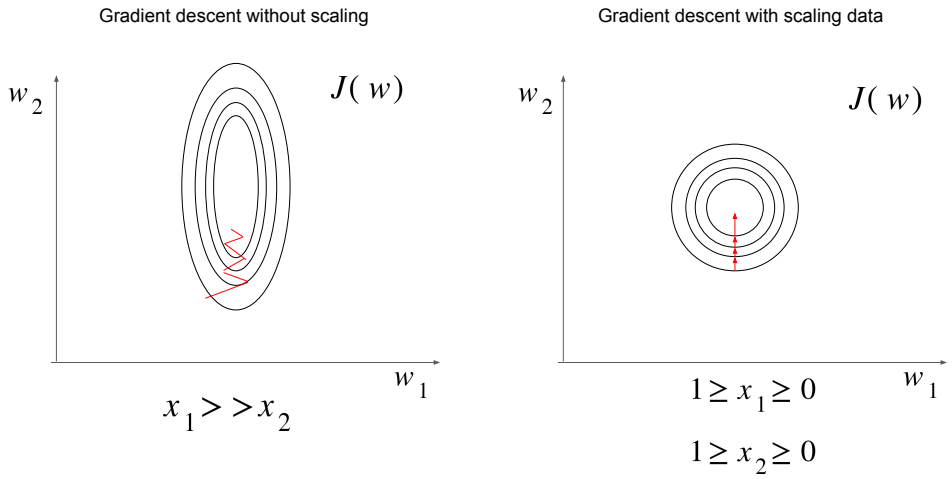


Figure 10: Loss function comparison for features scaling

normalization is given by the following equation,

$$X_{norm} = \frac{X - X_{Max}}{X_{Min} - X_{Max}}, \tag{3.6}$$

normalization scales the data equally between 0 and 1. The formula for standardization is as follows,

$$X_{std} = \frac{X - \mu_X}{\sigma_X}, \tag{3.7}$$

standardization scales the data into distribution with a mean and standard deviation of 1. In our experiments, the results with normalization or standardization are comparable, but they are far better than the results without data scaling.

3.2 Model for Snapshot Features

In Section 2.1.1, we describe the tracking problem into two objective functions. The first one is to find an ideal function, which can predict the most accurate result based on snapshot features. Such a concept allows us to consider this problem as an indoor localization problem. The localization application based on the learning method often uses snapshot features to predict the corresponding coordinates. The second objective function is to predict the coordinates of tracking objects using sequence data. In this chapter, we will introduce some methods of indoor tracking that are proposed for fingerprints.

doi:10.6342/NTU202203944

3.2.1 Support Vector Machine

The support vector machine (SVM) is the statistical learning-based supervised algorithm that separates two or more different clusters by finding a hyperplane. The goal of the SVM algorithm is to find a separating line that is as far apart as possible on the boundary so that the model is more resistant to noise. In general, the actual classification problem is more complex than two-dimensional, and it is not easy to find a linear answer. The kernel function can map the data to a higher-dimensional space and then do the partitioning.

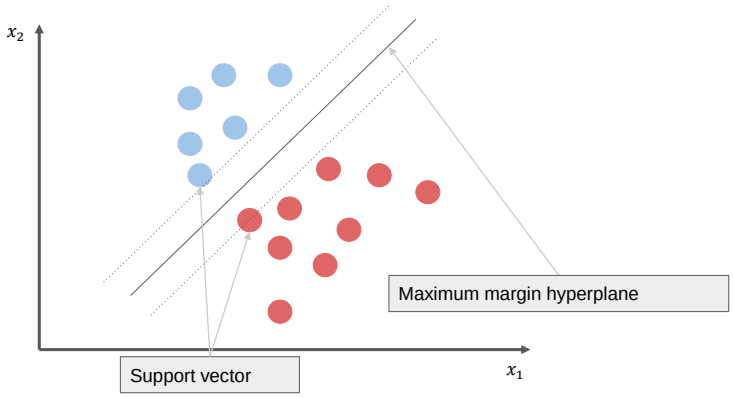


Figure 11: Support vector machine

3.2.2 K Nearest Neighbors Algorithm

The K-nearest neighbors algorithm (KNN) is an early learning method which is commonly used in learning-based localization applications (e.g., [21]). KNN algorithm contains two components. One is to find the k-nearest neighbors. As a result, the distance between the test feature and the reference features must be calculated. And the other is to determine the label of the test fingerprint by the votes of the k-nearest neighbors.

The fingerprint distance is defined as follows,

$$FD(TF, RF_i) = \frac{1}{n}\sum_{k=1}^{n} |v_k - v_k^i|,$$

where TF and RF^i are defined as $TF = [v_1, v_2, ..., v_n]$ and $RF^i = [v_1^i, v_1^2, ..., v_n^i]$. KNN algorithm can find out the nearest K reference points according to the fingerprint distance between test fingerprint and all reference fingerprints.In the case of KNN algorithm, the label of test fingerprint is decided by the vote of the nearest

doi:10.6342/NTU202203944

K RPs. In the case of weighted KNN, the weights are averaged according to the fingerprint distances between TF and RF, and the formula is as follows,

$$(x_p, y_p) = \frac{\sum_{i=1}^{k} \frac{1}{FD(TP,RP_i)}(x_i, y_i)}{\sum_{i=1}^{k} \frac{1}{FD(TP,RP_i)}},$$

where (x_i, y_i) are the coordinates of the i^{th} nearest RP.

3.2.3 Fully Connected Neural Network

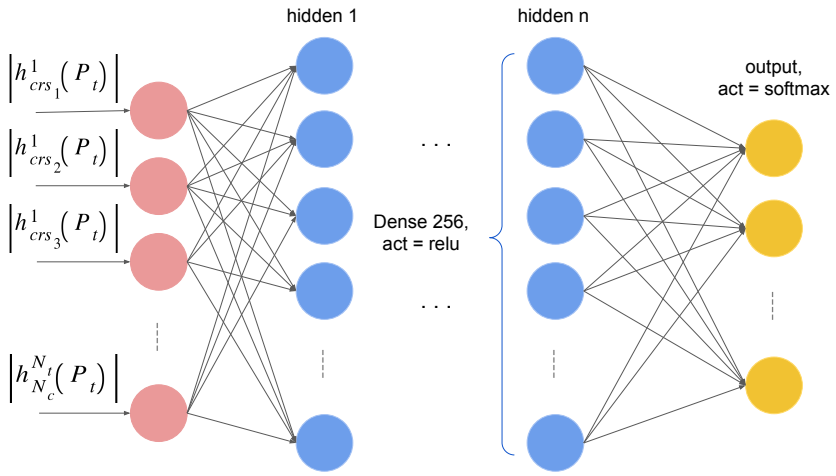


Figure 12: Architecture of fully connected neural network

Figure 12 is a multi-layer perceptron architecture, proposed in [16], using the snapshot LTE CSI vector $H(P_t)$ as the model input. The output of the SLN model is a weight vector $\omega = [w_1, w_2, w_3, ..., w_{N_{cell}}]$, where $w_i \in [0, 1), i = 1, 2, ..., N_{cell}$, satisfied the following condition, $\sum_{i=1}^{N_{cell}} w_i = 1$. In the SLN model, the rectified linear unit (Relu) is used as the activation function of the hidden layer and the softmax is used as the activation function of the model output layer.

The general solution for localization is directly picking up the reference point coordinates corresponding to the largest weight of model output as predicted results. To improve the accuracy of the neural network model, the combination of weights and reference point coordinates is used as a predicted result of the neural network model by the following equation,

$$\hat{p} = \sum_{i=1}^{N_{cell}} w_i * p_i, \tag{3.8}$$

where p_i is the coordinate of the i^{th} reference point. The SLN structure is shown in Table 2. We modify the architecture in the experiment in an attempt to improve

doi:10.6342/NTU202203944

layers	SLN
Input Layer	1×200
Layer 1	Dense 256 + Dropout 0.3, Activate function Relu
Layer 2	Dense 256 + Dropout 0.3, Activate function Relu
Layer 3	Dense 256 + Dropout 0.3, Activate function Relu
Layer 4	Dense 256 + Dropout 0.3, Activate function Relu
Output Layer	$1 \times N_{RPs}$, Activate function Softmax
Loss function	Cross Entropy

Table 2: Layer of SLN model

the accuracy of the FCNN model. Details of the experiment are given in section 3.4.4.

3.2.4 One-Dimensional Convolutional Neural Network

Before proposing our model for snapshot data, we would like to discuss the LTE CSI vector. Each element of the LTE CSI vector corresponds to the channel response of a subcarrier, so the vector is expressed as follows,

$$h(P_t) = [a_1 e^{-j2\pi f_1 \Delta t}, a_2 e^{-j2\pi f_2 \Delta t}, ..., a_{N_c} e^{-j2\pi f_{N_c} \Delta t}], \tag{3.9}$$

where f_i represents the frequency of the i^{th} subcarrier and a_i represents the amplitude of the i^{th} subcarrier which is received at the receiving end, and the vector conforms to the following inequality, $f_1 < f_2 < ... < f_{N_c}$. Then the free-space path loss formula is known as follows:

$$L(dB) = 20log_{10}(\frac{4\pi df}{c}), \tag{3.10}$$

where c represents the speed of light, d represents the propagation distance, and f represents the carrier frequency, from the formula we know that the frequency level and propagation distance will affect the power consumption. Ideal, for radio propagation, the low frequency level transmit with less loss power loss than hight frequency level, so it's know that $a_1 \geq a_2 \geq ... \geq a_{N_c}$ holds. In the actual case, there are usually some effects such as non-line of sight, interference, etc., and there are some patterns in the vector that are learned, such as the yellow line in Figure 13. The yellow line shows that the interference also affects the smoothness of the curve. In addition, the distance between UE and BS also affects the difference of amplitude between two neighboring subcarriers, mainly because the higher frequency subcarrier propagates over a long distance, resulting in more amplitude degradation, as in the case of the blue and red lines in Figure 13. Based

doi:10.6342/NTU202203944

layers	1DCNN
Input Layer	$1 \times 200 \times 1$
Layer 1	Conv1D(128, 6, 3, 'relu') + MaxPool1D(3, 1) + Flatten
Layer 2	Dense 512 + Dropout 0.3, Activate function ReLU
Layer 3	Dense 512 + Dropout 0.3, Activate function ReLU
Layer 4	Dense 512 + Dropout 0.3, Activate function ReLU
Layer 5	Dense 512 + Dropout 0.3, Activate function ReLU
Output Layer	$1 \times N_{RPs}$, Activate function Softmax
Loss function	Cross Entropy

Table 3: CNN model architectures

on these analyses, we propose the one-dimensional Convolutional Neural Network (1DCNN) model for the LTE CSI vector.

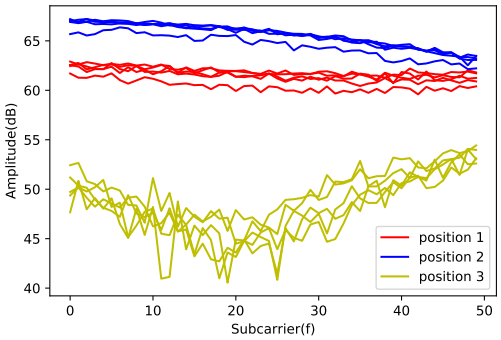


Figure 13: The 1^{st} to 50^{th} real part of LTE CSI in three collection positions

Figure 14 shows a typical 1D CNN model, which is divided into two parts, one for feature extraction and the other one for classification or simulation functions. The feature extraction model consists of a convolutional layer, a pooling layer, and a flatten layer. The main function of the convolutional layer is to extract useful features by using a filter, and the convolutional layer can reduce noise and achieve a sharpening effect. The flatten layer is used for data shape transformation. In the rest of the model, the fully connected network (FCN) is used. In the 1DCNN feature extraction model, we mainly consider the classification problem, so the output uses softmax as the activation function, and the whole model uses cross entropy as the loss function. The rest of the FCN uses ReLU as the activation function. We call a 1-dimensional CNN model with one convolutional layer as 1D1LCNN and a 1-dimensional CNN with two convolutional layers as 1D2LCNN. We will experiment whether 1DCNN increasing the number of convolutional layers can increase the accuracy of the model.

doi:10.6342/NTU202203944

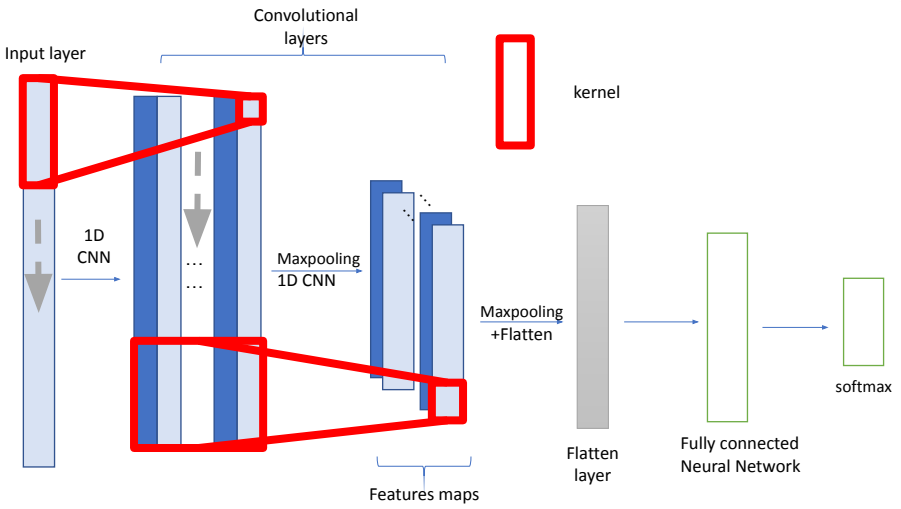


Figure 14: 1D Convolutional neural networ

layers	MICNN	
Input Layer	$1 \times 200 \times 1$	$1 \times 199 \times 1$
layer	Conv1D(128, 6, 3, 'relu')	Conv1D(64, 6, 3, 'relu')
layer	MaxPool1D(3, 1)	MaxPool1D(3, 1)
layer	Flatten	Flatten
layer	Dense 512 + Dropout 0.3, Activate function ReLU	Dense 256 + Dropout 0.3, Activate function ReLU
concatenate layer		
Layer	Dense 512 + Dropout 0.3, Activate function ReLU	
Layer	Dense 512 + Dropout 0.3, Activate function ReLU	
Layer	Dense 512 + Dropout 0.3, Activate function ReLU	
Output Layer	$1 \times N_{RPs}$, Activate function Softmax	
Loss function	Cross Entropy	

Table 4: Layers of Multi-Input CNN model

3.2.5 Proposed Model

To conclude the previous discussion in section 3.1.2, we use two types of vectors, one is the vector which is composed of the real part of CSI, here called the real vector, and the other is the phase difference vector of CSI, here called the phase difference vector. The real number vectors are 1×200 dimension vectors, and the phase difference vectors are 1×199 vectors. We designed a model to obtain the better performance with these two kinds of vector than the model with real vectors. The architecture of the model is shown in Figure 15, and Table 4 shows the parameters of the MICNN.

3.3 Time Domain Data Fusing method

While in the previous section, we explored models for extracting features from data. In this section, we explore methods that can combine sequence features which is extracted by the model to do indoor tracking.

doi:10.6342/NTU202203944

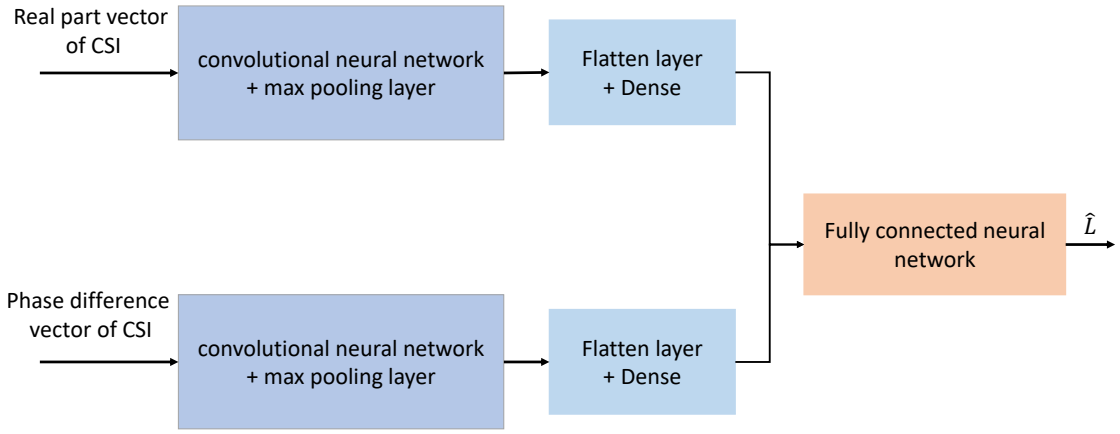


Figure 15: Architecture of the multi-input convolutional neural network

3.3.1 Fusion Network

Fusion network (FN) is mainly proposed in [16], which is mainly considered for indoor localization applications, as shown in Figure 16. Although this network is proposed for indoor localization applications, we believe that when the data meet some specific conditions, the features with position change in adjacent time points can help to predict the current position P_t. The prediction obtained from the output of the extraction model is written as the following equation,

$$\hat{P}_t = \hat{h}(v_t), \tag{3.11}$$

where v_t represent the LTE CSI vector at time point t, the extraction model is considered as a function $\hat{h}()$ whose output is the coordinates. And we know from Equation (2.1) that we can write the relationship between P_t and P_{t-1} as the following equation,

$$P_t = P_{t-1} + \Delta P_t, \tag{3.12}$$

where ΔP_t is a variable value that depends on the movement,this equation allows us to find the relationship between P_{t-k} and P_t. Under special conditions, ΔP_t is considered negligible. First, the object does not have any movement. Second, when the sampling is frequent, the sampling interval is very short, the object movement does not change much. Third, the object itself does not move fast.

doi:10.6342/NTU202203944

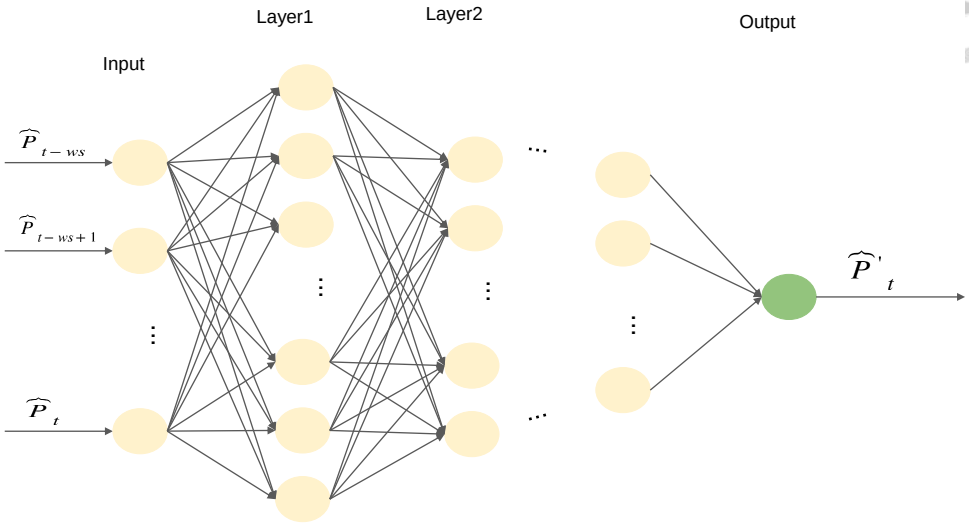


Figure 16: Fusion network

layers	FN
Input Layer	$length \times 2$
Layer 1	Dense 100, Activate function ReLU
Layer 2	Dense 64, Activate function ReLU
Layer 3	Dense 48, Activate function ReLU
Layer 4	Dense 12, Activate function ReLU
Output Layer	1×2, Activate function Linear
Loss function	Mean square error

Table 5: Layer of fusion network

doi:10.6342/NTU202203944

3.3.2 Recurrent Neural Networks

We use two types of RNN models respectively. The first one is shown in Figure 17, which is a MISO model that combines the RNN outputs to predict the position, the LSTM cell uses TanH function as the activation function, the FCN uses ReLU as the activation function, and the loss function is the mean square error. This model is called SIMO RNN.

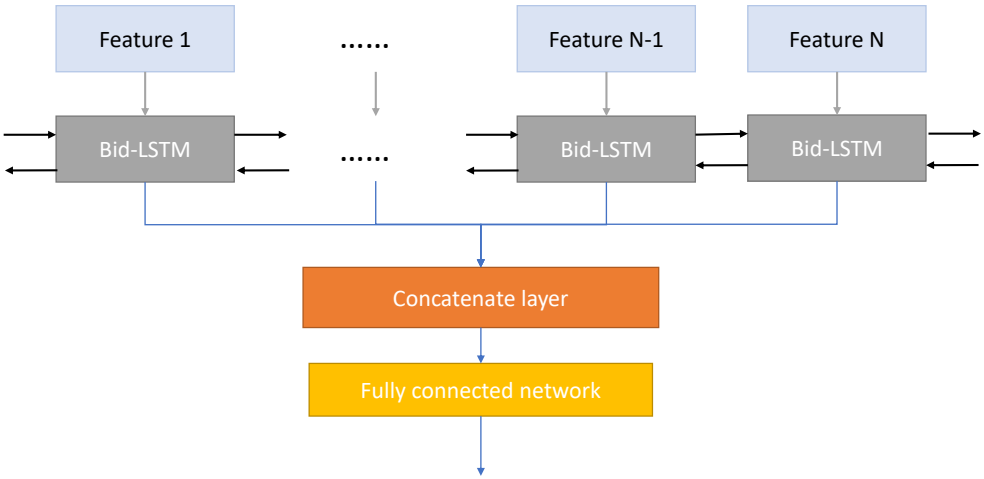


Figure 17: Architecture of MISO LSTM model

The second model is shown in Figure 18. It is a MIMO model in which the output of the LSTM cell will be calculated by FCN to find the position coordinates. The second model has the same activate function settings as the first one, but the difference is that the loss function uses the mean square error for each data output. This model is called MIMO RNN. The detailed parameters of the two models are shown in Table 6.

There are some differences between MIMO RNN and MISO RNN. MISO RNN is using all the timing outputs to predict the output, so the positive timing input and the inverse timing input helps predict the output of the last timing, while the MIMO model using the bidirectional model is not very helpful because the hidden state is zero, representing the inverse timing RNN has no predecessor state to help predict.

doi:10.6342/NTU202203944

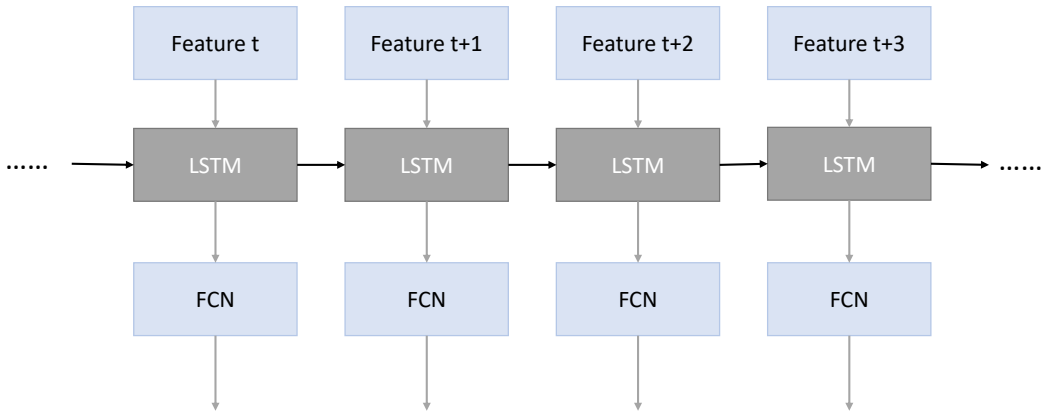


Figure 18: Architecture of MIMO LSTM model

layers	MISO RNN
Input Layer	Input shape = (batch size, length, feature size)
Layer 1	bidirectional-LSTM(100,layers = 1, dropout = 0.2)
Layer 2	Flatten layer
Layer 3	Dense 512 + Dropout 0.3, Activate function ReLU
Layer 4	Dense 512 + Dropout 0.3, Activate function ReLU
Layer 5	Dense 512 + Dropout 0.3, Activate function ReLU
Layer 6	Dense 512 + Dropout 0.3, Activate function ReLU
Output Layer	output shape = (batch size, 2)
Loss function	Mean square error
layers	MIMO RNN
Input Layer	Input shape = (batch size, length, feature size)
Layer 1	LSTM(256, layers = 1, dropout = 0.2)
Layer 2	Dense 512 + Dropout 0.3, Activate function Relu
Output Layer	output shape = (batch size, length, 2)
Loss function	Mean square error

Table 6: Layer of SIMO and MIMO RNN

doi:10.6342/NTU202203944

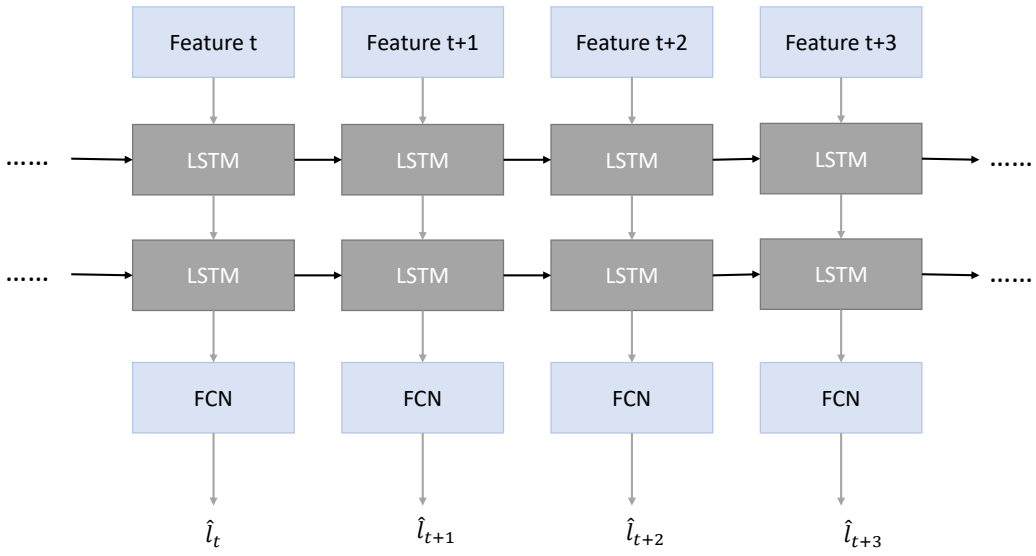


Figure 19: Architecture of stack RNN

layers	Stack RNN
Input Layer	Input shape = (batch size, length, feature size)
Layer 1	LSTM(256, layers = 2, dropout = 0.2)
Layer 2	Concatenate layer
Layer 3	Dense 512 + Dropout 0.3, Activate function ReLU
Output Layer	output shape = (batch size, 2)
Loss function	Mean square error

Table 7: Layer of stack RNN

3.3.3 Stack RNN

The stack RNN is different from the original RNN in that the stack RNN refers to an RNN with two or more RNN layers, and the RNN of the upper layer will use the output as the input of the RNN of the lower layer. The architecture is shown in Figure 19.

3.3.4 DL-RNN

The DL-RNN model is proposed in [15] and the structure of DL-RNN is similar to that of stack RNN. The DL-RNN model is divided into Location matching RNN and Location filtering RNN. Location matching RNN is used to learn rough mapping relationships between fingerprint and location, and location filtering RNN is used to obtain a more accurate location. The loss function of DL-RNN is shown

doi:10.6342/NTU202203944

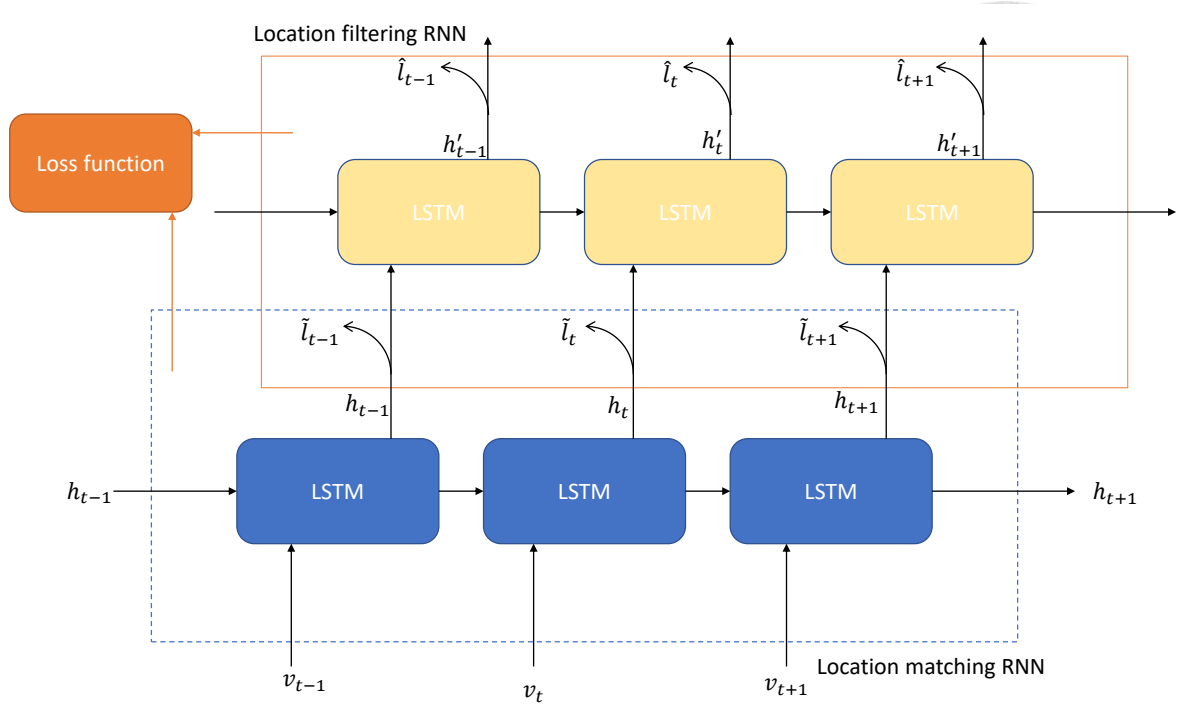


Figure 20: DL-RNN

layers	DL-RNN
Input Layer	Input shape = (batch size, length, feature size)
part of RNN	LSTM(256, layers = 1, dropout = 0.2)
part of FCNN	Dense 256 + Dropout 0.3, Activate function ReLU
Output Layer	output shape = (batch size, 2)
Loss function	Weight combination of MSE

Table 8: Layer of DL-RNN

as follows,

$$\lambda_1 \sum_{t=1}^{S} ||\hat{l}_t - l_t|| + \lambda_2 \sum_{t=1}^{S} ||\tilde{l}_t - l_t||,$$

where $||.||$ represents the Euclidean distance, the loss function has two parts, and the weights of these two parts in the loss function are $\lambda_1 + \lambda_2 = 1$.

3.4 Evaluation Results

3.4.1 Evaluation Criteria

Common criterions for evaluating localization accuracy are: Mean distance error (MDE) (3.13), Mean Square Error (MSE) (3.14), Root Mean Square Error (RMSE) (3.15), and Median Localization Error (MLE) (the 50^{th} of the CDF).

doi:10.6342/NTU202203944

$$MDE = \frac{1}{N_s}\sum_{k=1}^{N_s}||P_{k,truth} - P_{k,pred}||, \tag{3.13}$$

$$MSE = \frac{1}{N_s}\sum_{k=1}^{N_s}||P_{k,truth} - P_{k,pred}||^2, \tag{3.14}$$

$$RMSE = \sqrt{\frac{1}{N_s}\sum_{k=1}^{N_s}||P_{k,truth} - P_{k,pred}||^2}, \tag{3.15}$$

where N_s represents the total number of positions predicted from all features. We also use the cumulative distribution function (CDF) to evaluate the model. Sometimes, the CDFs are too similar, it is difficult to evaluate the models. The 25^{th}, 50^{th}, 75^{th}, 100^{th} percentile values can be a good description of the CDF. As a result, we also use different percentile of the CDF to evaluate the performance.

3.4.2 Cross Validation Method

Machine learning models are likely to have different biases depending on the training set, and using a specific training set to evaluate the performance of the model is not objective enough. We use a similar k-fold cross-validation method to validate our models, where we view the data collected by turtlebot3 on a single trajectory as a trajectory dataset since the data collected in the different trajectories have less correlation. As shown in Figure 21, we use 8 datasets in Experiment 1 to evaluate the performance. Depending on the different trajectory datasets used as training sets, we can obtain validation metrics E_i (like MDE, RMSE). We evaluate the model by averaging the metrics of experiments with the same number of the trajectory dataset. This approach can also be used to evaluate the improvement of model performance when the amount of trajectory dataset increases.

3.4.3 Platform

We use a Mac pro as the LTE base station server and USRP B210 as the signal transceiver platform. We run OpenAirInterface (OAI) BS on the base station, as shown in Figure 22(a). About LTE transmission configuration, we set 25 resource blocks and a single antenna port. We used turtlebot3 waffle pi(abbreviated as turtlebot3) as a roamer as shown in Figure 22(b). However, the performance of the original single-board computer, Raspberry Pi, was not enough to run OAI UE, so we use another single-board computer with better performance, up board. In addition, the turtlebot3 is equipped with various high precision sensor modules, including a 3-axis magnetometer, 3-axis accelerometer, 3-axis gyroscope, and smart motors, which can make the SLAM algorithm more accurate. Our motion

doi:10.6342/NTU202203944

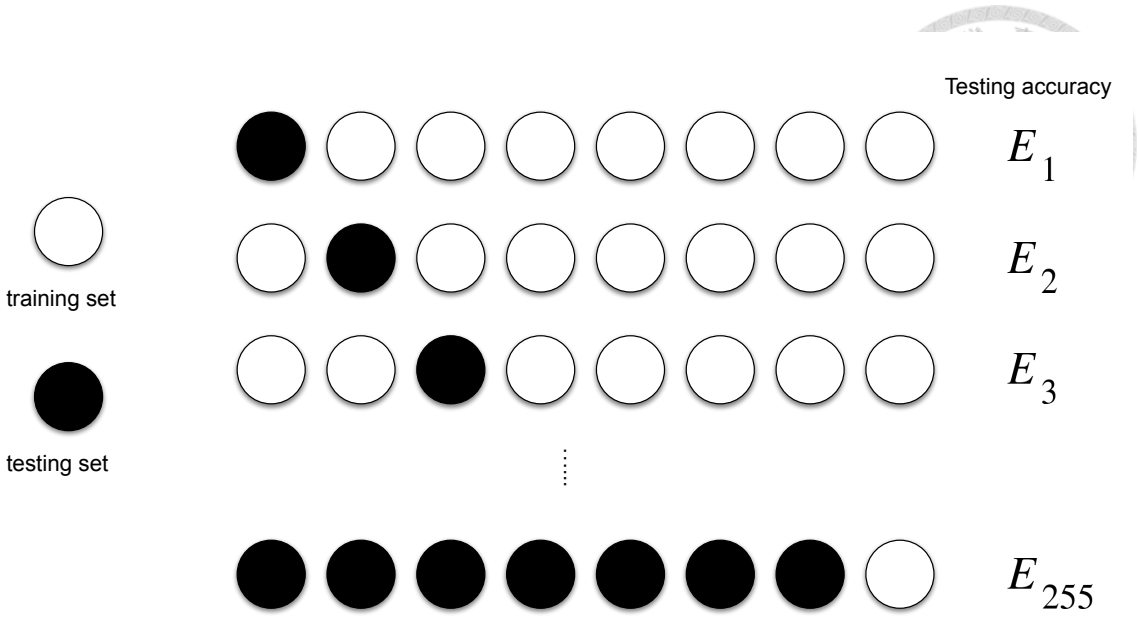
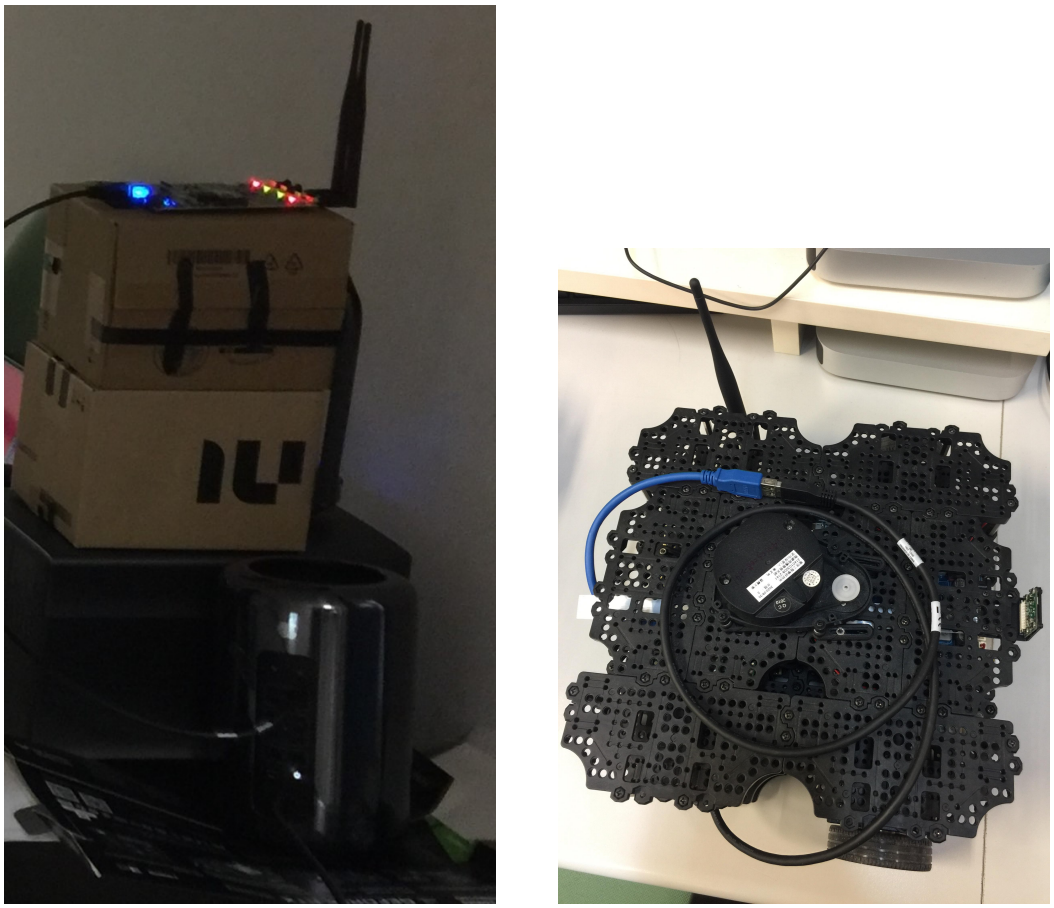


Figure 21: 8-fold cross validation



(a) LTE Base station (b) Turtlobot3

Figure 22: (a) is a base station and (b) is a roamer

doi:10.6342/NTU202203944

default parameter	
Resource blocks	25
Number of antenna port	1
Velocity (static)	0.0 m/s
Velocity (Dynamic)	<0.1 m/s

Table 9: Default parameter

data is collected by the smart motor, and the speed is controlled within 0.1m/s when collecting data. To run OAI UE, turtlebot3 is also equipped with a USRP B210 to extract CSI features. To sum up, the parameters are set as in Table 9.

3.4.4 Experimental environment one

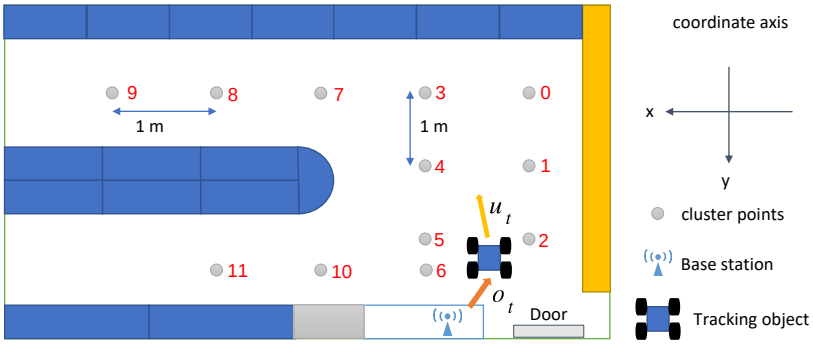


Figure 23: Experimental scenario

The experiment was conducted in BL521, and we placed the BS on the top of the cabinet. We use Wi-Fi router to receive LTE CSI transmitted from the turtlebot3. We marked some cluster points on the actual map, we collected the motion data and LTE CSI vector when walking, assigning data to nearby cluster points is helpful for model training. Figure 24 shows the trajectories for data collection, where the coordinates are used as ground truth which is estimated by the ROS SLAM algorithm and corrected by ourselves. The SLAM algorithm used for longer walking distance will lead to greater estimation error, so our robot did not walk more than 20 meters in the path for data collection, this limit can keep the ground truth error within $5cm$. Each path is an independent experiment, the robot collects data while walking, regarded as a dataset, each dataset contains 3500 6000 pieces of snapshot data, each piece of data contains a ground truth

doi:10.6342/NTU202203944

data quantity	
Trajectory 1	4993
Trajectory 2	4082
Trajectory 3	5698
Trajectory 4	4406
Trajectory 5	5701
Trajectory 6	5050
Trajectory 7	4701
Trajectory 8	3648

Table 10: The number of data in each dataset

Model	MDE (1 path)	MDE (2 paths)	MDE (4 paths)	MDE (7 paths)
SVM	1.745	1.711	1.645	1.590
KNN	1.504	1.475	1.447	1.426
WKNN	1.486	1.459	1.430	1.406
SLN	1.410	1.389	1.367	1.352
1D2LCNN	1.407	1.357	1.286	1.231
1D1LCNN	1.403	1.333	1.253	1.191

Table 11: Comparison of different extraction models

coordinate, motor speeds, LTE CSI vector.

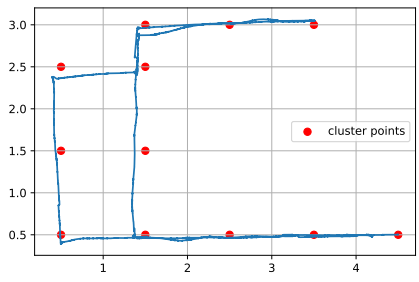
3.4.5 Feature Extraction Model Comparison

We compared various models for snapshot data, mainly using two kinds of criteria, and our results are shown in Figure 25 and Table 11. In terms of results, the performances of the neural networks are about the same when the number of training sets is small. But neural network method is better than traditional approaches such as SVM, WKNN, and KNN by at least five percent. When the number of training sets was increased to seven, the performance of 1D1LCNN improved by 15%, and the performance of 1D1LCNN is 12% better than the performance of SLN when the number of trajectory data sets increased to seven.

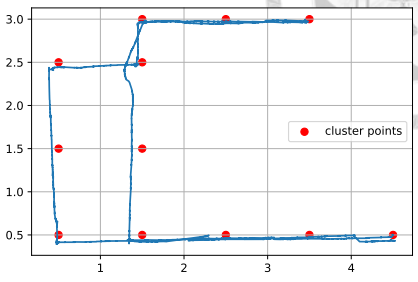
3.4.6 Loss Function Comparison

We try to use mean square error as the loss function to obtain the better performance. The results are shown in Figures 26 and Table 12. We use SLN and 1D1LCNN models to experiment. We observe that the results of the SLN model and 1D1LCNN model are very different. The classification SLN model can learn more features compared to the regression SLN model. When the amount of training datasets are small, the performance of the regression 1D1LCNN model is

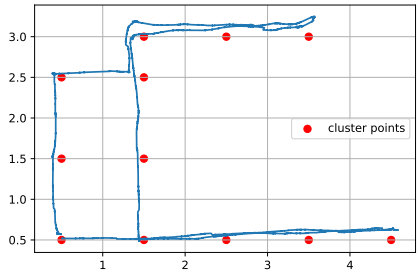
doi:10.6342/NTU202203944



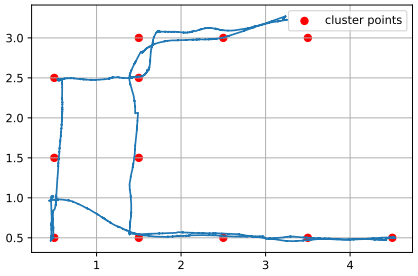
(a) trajectory 1



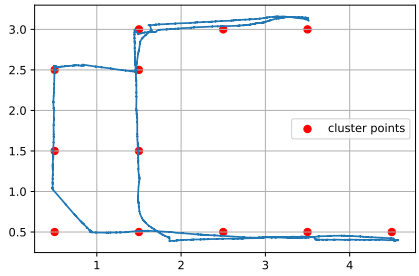
(b) trajectory 2



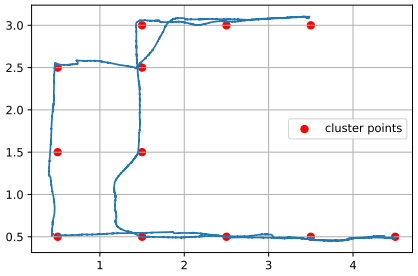
(c) trajectory 3



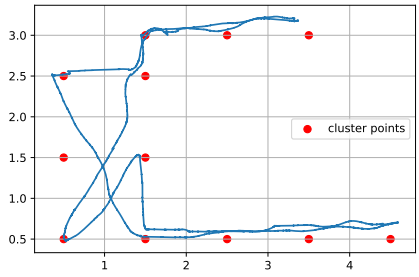
(d) trajectory 4



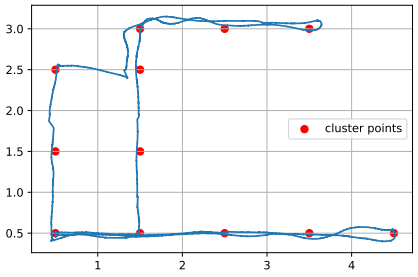
(e) trajectory 5



(f) trajectory 6



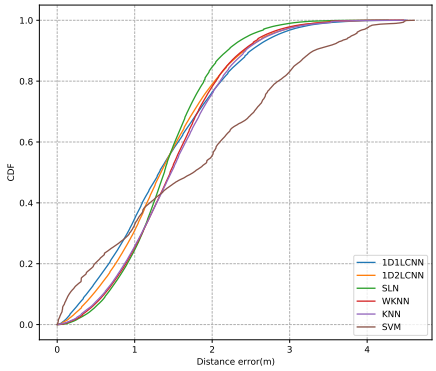
(g) trajectory 7



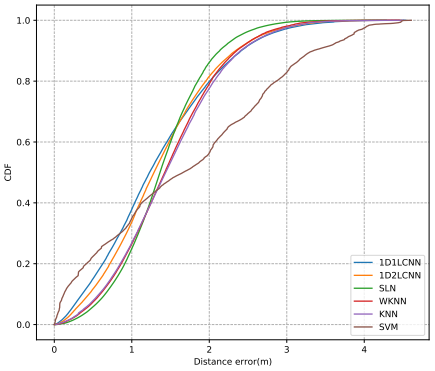
(h) trajectory 8

Figure 24: The trajectories of data collection

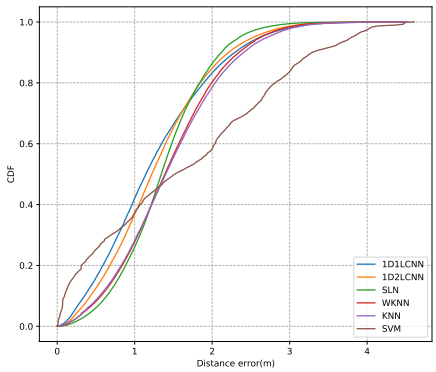
doi:10.6342/NTU202203944



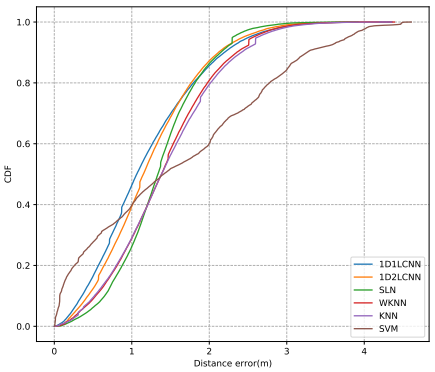
(a) 1 trajectory data set



(b) 2 trajectory data sets



(c) 4 trajectory data sets



(d) Different models performance trained on 7 trajectory data sets

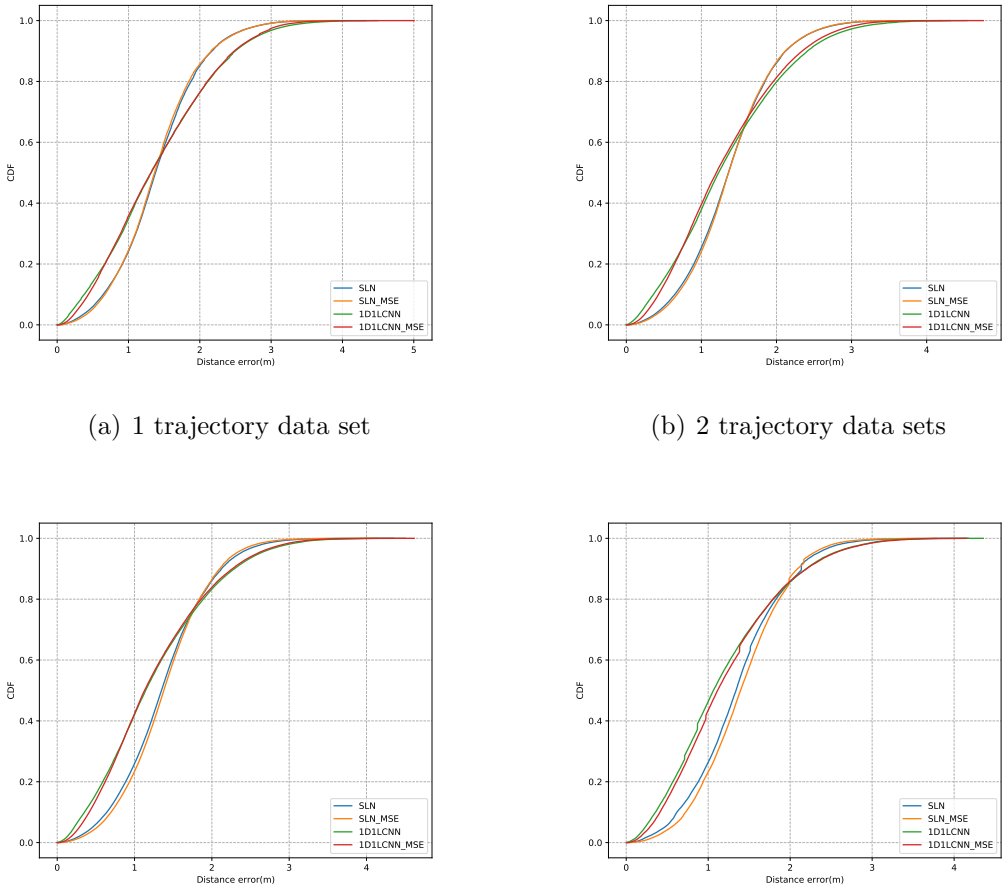
Figure 25: CDF of model

doi:10.6342/NTU202203944

Model	MDE (1 path)	MDE (2 paths)	MDE (4 paths)	MDE (7 paths)
SLN	1.405	1.383	1.372	1.354
SLN (MSE)	1.398	1.391	1.393	1.399
1D1LCNN	1.403	1.333	1.253	1.191
1D1LCNN (MSE)	1.402	1.309	1.253	1.224

Table 12: Comparison of the models with MSE and cross entropy

better than the performance of the classification 1D1LCNN model; when the number of training datasets is increased to seven, the performance of the classification 1D1LCNN model can obtain better performance.



(a) 1 trajectory data set

(b) 2 trajectory data sets

(c) 4 trajectory data sets

(d) 7 trajectory data sets

Figure 26: CDF of different model with MSE and cross entropy

3.4.7 Label Smoothing Method

To avoid overfitting of the ML model, we try to do label smoothing. We use the SLN model to test whether the label smoothing method is feasible. The label of the original training set uses one-hot encoding to label the data. We defined

doi:10.6342/NTU202203944

Model	MDE (1 path)	MDE (2 paths)	MDE (4 paths)	MDE (7 paths)
1D1LCNN	1.403	1.333	1.253	1.191
1D1LCNN(smoothing)	1.378	1.311	1.240	1.211

Table 13: Comparison of different label methods

12 reference points in the field, and the definition method is to categorize the points as the nearest reference point, this method is called one-hot encoding. We defined the weights $\frac{e^{\frac{-d_j}{\lambda}}}{\sum_{i=1}^{12} e^{\frac{-d_i}{\lambda}}}$ based on the distance between the test points and the j^{th} reference points, this method is our label smoothing. Our results are shown in Figure 27 and Table 13. Our results show that with less training data, the model performance with label smoothing is slightly improved. As the training data increases, the model performance with label smoothing is slightly worse.

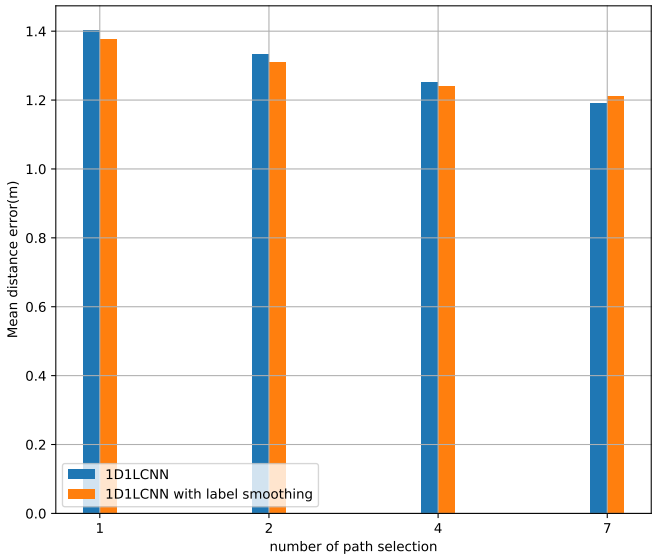


Figure 27: Comparison the average MDE of model with or without label smoothing

3.4.8 Considering the Phase Difference as Model Input

In this subsection, we compare the model perfomrance for using phase difference. We compare the performance of MICNN, 1D1LCNN, and SLN, and Figure 28 and Table 14 show the results. Regardless of the data amount of training data, the MI-CNN model performs slightly better than other models. The performance of the MI-CNN model is 12.8% better than the performance of the SLN model. From the analysis of the importance of the labels in Section 3.1.2, the phase difference can provide information on the distance between the data collection site

doi:10.6342/NTU202203944

Model	MDE (1 path)	MDE (2 paths)	MDE (4 paths)	MDE (7 paths)
MICNN	1.395	1.288	1.222	1.182
1D1LCNN	1.403	1.333	1.253	1.191
SLN	1.410	1.389	1.368	1.352

Table 14: Comparing the average MDE of model with or without phase difference

and the base station. Therefore, the phase difference for BL521 can provide less information. The phase difference can provide more information for a narrow area.

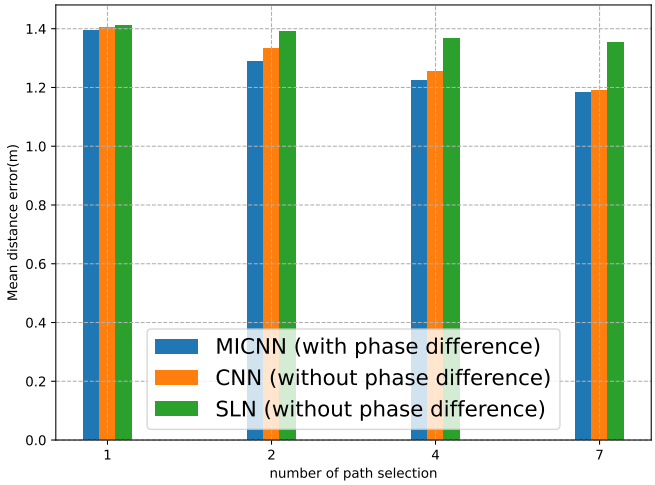


Figure 28: Comparison the average MDE of model with or without phase difference

3.4.9 Cascaded Model Comparison

We have tried several different combinations of cascaded models. We first compare the MDE performance of different Cascaded models, and the results are shown in Figure 29 and Table 15. These data fusion models have been described in detail in Section 3.3. The SLN + FN model is the cascaded model proposed in the paper [21]. We combine SLN+RNN models to form a cascaded model, and the performance of SLN+RNN is no better than the performance of SLN+FN models. This result may be due to the low accuracy of SLN and poor generalization of RNN. For MDE, all the CNN-RNN-related cascaded models have similar performance. For MDE, MICNN+RNN has the best performance compared to other cascaded models when the training data contains 2,4,7 trajectory datasets.

The comparison of the average MDE is not yet able to fully evaluate the performance of the cascaded models. As a result, We use root mean square (RMSE) to evaluate the performance of different cascaded models. The results are shown

doi:10.6342/NTU202203944

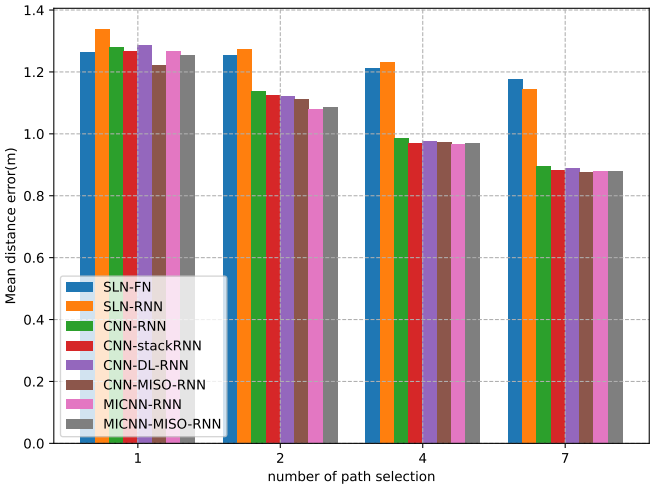


Figure 29: Comparing the average MDE of the different cascaded models

Model	MDE (1 path)	MDE (2 paths)	MDE (4 paths)	MDE (7 paths)
SLN-FN	1.262	1.255	1.211	1.176
SLN-RNN	1.339	1.273	1.231	1.146
CNN-RNN	1.28	1.139	0.987	0.894
CNN-stackRNN	1.267	1.123	0.971	0.881
CNN-DL-RNN	1.285	1.122	0.976	0.89
CNN-MISO-RNN	1.222	1.111	0.973	0.877
MICNN-RNN	1.268	1.079	0.967	0.879
MICNN-MISO-RNN	1.254	1.087	0.97	0.879

Table 15: Comparing the Average MDE of Cascaded Models

doi:10.6342/NTU202203944

Model	RMSE (1 path)	RMSE (2 paths)	RMSE (4 paths)	RMSE (7 paths)
SLN-FN	1.391	1.393	1.346	1.303
SLN-RNN	1.51	1.45	1.409	1.288
CNN-RNN	1.516	1.365	1.205	1.1
CNN-stackRNN	1.52	1.366	1.204	1.103
CNN-DL-RNN	1.547	1.363	1.206	1.108
CNN-MISO-RNN	1.425	1.325	1.189	1.094
MICNN-RNN	1.453	1.275	1.175	1.077
MICNN-MISO-RNN	1.425	1.281	1.185	1.093

Table 16: Comparing the average RMSE of cascaded models

in Figure 30 and Table 16. For RMSE, the MICNN+RNN performs slightly better than other cascaded models.

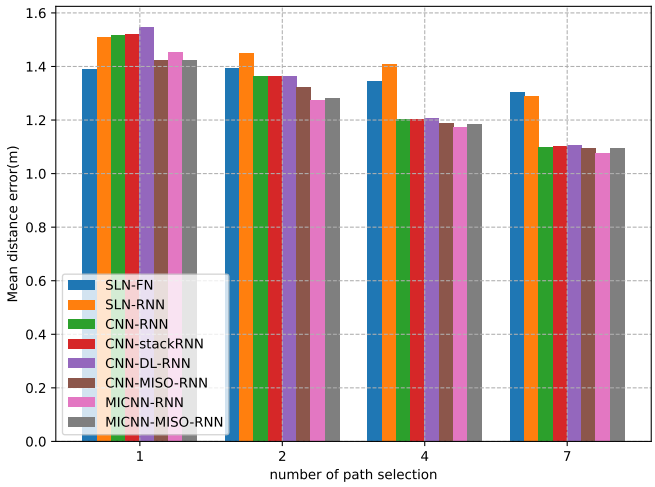


Figure 30: Comparing the average RMSE of the different cascaded models

3.4.10 Cascaded Models Comparison for Different Input Features

The results are shown in Figure 31 and Table 17. We used the predicted position as input for RNN or FN in the previous subsection, and we used the second layer output of CNN mentioned in Section 3.2.4 as input for RNN with the input shape (Batch size, length size, 512). We use MIMO RNN to compare the performance based on two input features. From the results, the performances of using two different features as RNN inputs do not differ much. This result may be because there is not much information left in the output from the middle layer of the CNN model, which contains many 0 elements. In our experiments, using the middle output of the 1D1LCNN model as RNN input does not necessarily lead to better performance, more often using the predicted location as RNN input can

doi:10.6342/NTU202203944

Model	MDE (1 path)	MDE (2 paths)	MDE (4 paths)	MDE (7 paths)
CNN-RNN(2)	1.28	1.12	0.975	0.884
CNN-RNN(512)	1.301	1.145	0.995	0.866

Table 17: Comparison of Different Input Features for the Cascaded models

lead to better model performance.

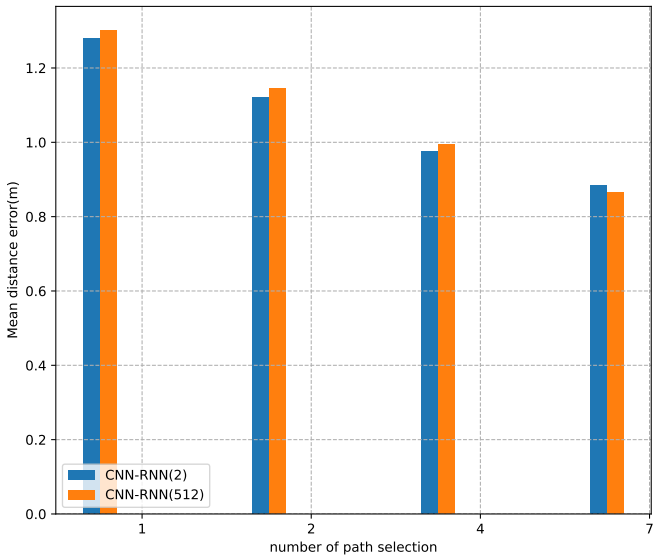


Figure 31: Comparison of performance for different RNN input features

3.5 Summary

According to the results of our experiments, we judge the merits of the model mainly on the condition that the training set contains 2, 4, and 7 traces. The MICNN+RNN is the combination that perform better than other combinations in our experiments. The MICNN+RNN obtain good performance in the training dataset including 2, 4, and 7 trajectory datasets. The performance of MICNN+RNN is better than most of the 1D1LCNN with RNN-related models when training datasets including 2, 4, and 7 trajectory datasets.

doi:10.6342/NTU202203944

CHAPTER 4

LOCALIZATION DEPLOYMENT OF MODEL PERFORMANCE IMPROVEMENT

Two research fields can achieve better model performance with less training data. One is data augmentation. The other is transfer learning, where learned knowledge of the source model is transferred to the target model. With the development of the generative model, there have been many studies on using generative models to improve the performance of judgment models in other applications. In this chapter, our goal is to improve the performance of the feature extraction model. The time domain data fused model is prone to overfit, and the input features of the data fused model are simple. The system accuracy still depends on the prediction of coordinates by the feature extraction model.

4.1 Data Augmentation

We first applied generative models to improve the performance of the feature extraction model, and several papers [32], [33], [34] have been published to show the feasibility of generative models. Therefore, we try to apply GAN and VAE in our system.

4.1.1 Problem and Viewpoints

Data augmentation is a technique that uses limited data to generate more equivalent data to expand the training data set. Data augmentation is an effective means to overcome the shortage of training data and is currently used in deep learning for image recognition, speech, etc. However, the difference between the generated data and the real data inevitably brings the problem of noise.

In traditional artificial intelligence applications, data augmentation is commonly applied to image recognition applications. Image data can be expanded with training data by flipping left and right, offsetting, etc. In recent years, the development of generative models such as variational auto-encoder [35] (VAE), generative adversarial network [36] (GAN), generative stochastic networks [37] (GSN), Boltzmann machine [38], and other techniques have been proposed, and data augmentation is easily applied to other fields. This section, we will focus on GAN and VAE for feature generation.

doi:10.6342/NTU202203944

4.1.2 Generative Adversarial Network

The algorithm of basic Generative adversarial network (GAN) [36] is described in Section 2.4.2, but the data generated by basic GAN are random. Basic GAN cannot generate data with specific features. For this reason, using GAN to enhance the data is a technique. Some papers use the data generated by the generative model for semi-supervised learning of the model, pseudo labeling of fake data, the penalty for loss function of the identified model using fake data, etc. Data augmentation is performed using GANs that can generate specific conditions, such as CGAN [39], infoGAN [40], and ACGAN [41]. However, these GANs do not help the performance of localization model. The accuracy of our localization models is usually not higher than 50%, so the discriminator cannot fully learn the features that are useful for localization.

We have tried various approaches to train the generative model, and it is helpful to obtain the generative model that can generate the data with specific labels. This approach is proposed in [42] and [32], and Figure 32 shows the method. By dividing the data by different labels as training data sets and training the generative model by the data with the specific label, this method can obtain the simulation data with the label. We also try to train a generative model for all domain data without classification, and this method cannot distinguish what labels the generated data belong. Besides, the generative model is prone to mode collapse. In our thesis, we also tried to use 1DCNN model as the generative model, but the result is not satisfactory, so we adopt the similar DNN model proposed in [42] to generate fake data.

4.1.3 Variational Auto-encoder

Variational auto-encoder (VAE) is a generative model proposed in [35], and Figure 33 is the architecture of VAE. VAE can be divided into two models, the encoder model and the decoder model. The Encoder model of VAE maps the inputs to the mean and standard deviation of Gaussian distribution. The noise of the Gaussian distribution combined with the additional input can form a latent variable as the input to the decoder. This method will make the image generated by VAE slightly different from the original image, which may be a change in the feature, angle, or hue.

The loss function of VAE includes KL divergence and reconstruction loss, which are expressed as follows:

$$L(\theta, x_i) \simeq \frac{1}{2}\sum_{j=1}^{J}\left(exp(\sigma_j^{(i)}) - (1+\sigma_j^{(i)}) + (\mu_j^{(i)})^2\right) + ||x_{rec}^{(i)} - x_i||^2, \tag{4.1}$$

doi:10.6342/NTU202203944

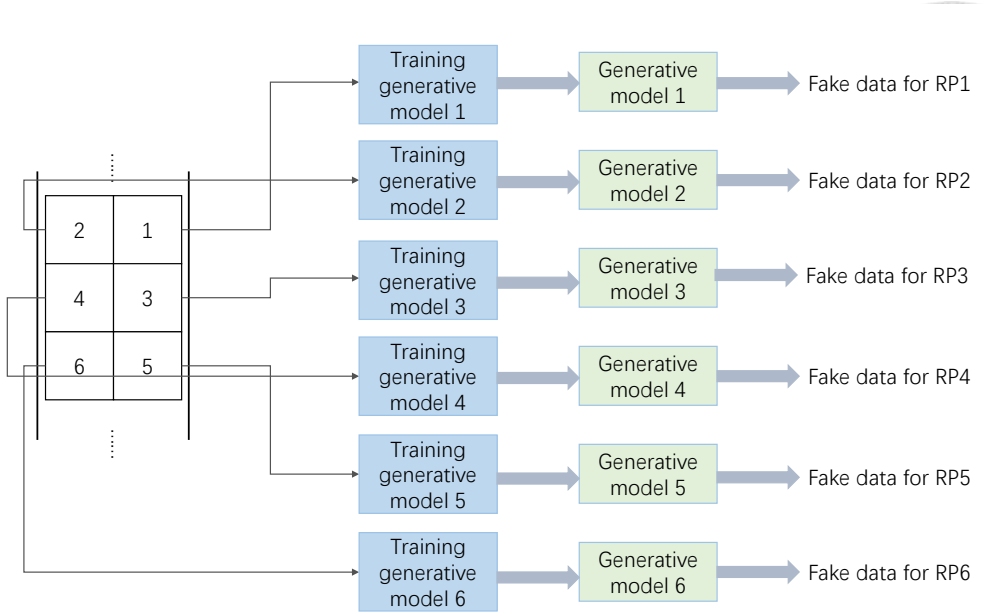


Figure 32: Data selection approach for training generative model

where $||x_{rec}^{(i)} - x_i||^2$ is the reconstruction loss, and the rest of loss function $L(\theta, x_i)$ is the KL divergence. In this thesis, we also divide the data into multiple training sets according to labels to train the VAE model to avoid generating data with random labels.

4.2 Transfer Learning Methods

In our experiments, using data augmentation is not effective in improving the performance of feature extraction models, so we try to apply parameter-based transfer learning to improve the model performance.

4.2.1 Model Fine-tuning

The concept of model fine-tuning is shown in Figure 34. Model fine-tuning is a common transfer learning method for labeled target domain data. The source domain overlaps with the target domain in terms of distribution. The source domain data is sufficient and complete, while the target domain is the target task and target domain data is not sufficient. The model fine-tuning method uses the model trained from sufficient data to train the target model.

In many applications, the labeled data is valuable and limited. For image applications, using a pre-trained CNN model as the initial model to go down for model fine-tuning can solve the problem that a small amount of data is difficult to train a good model and filter. In the first case, when the pre-trained model already has a strong recognition capability, the target and source domains have

doi:10.6342/NTU202203944

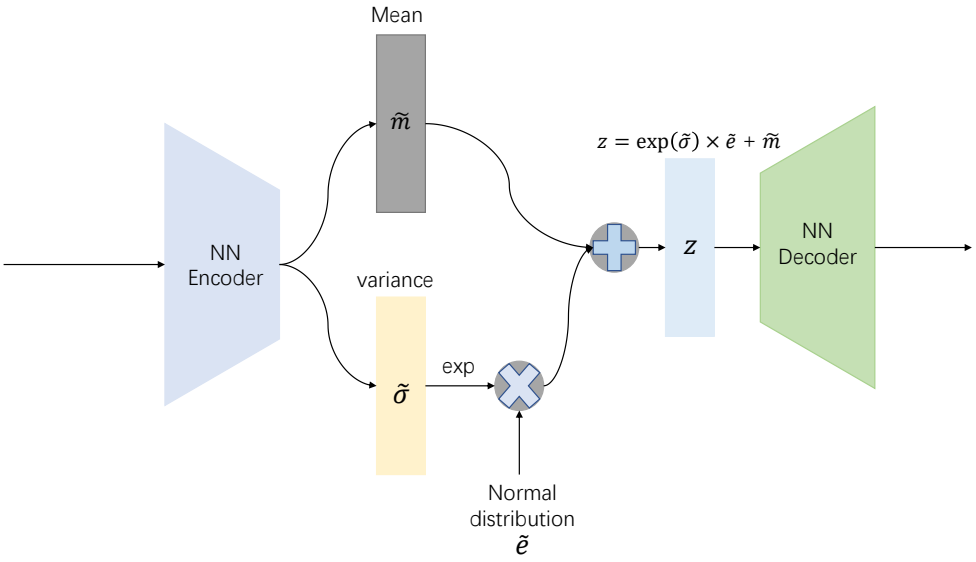


Figure 33: Architecture of Variational auto-encoder

Model	Encoder	
Input layer	$BatchSize \times 399$	
Layer 1	Dense 512 + activate function ReLU	
Layer 2	Dense 256 + activate function ReLU	
Layer 3	Dense 128 + activate function ReLU	
Output layer	Mean laten : Dense 16 + activate function Linear	Variance laten : Dense 16 + activate function Linear
Model	Decoder	
Input layer	$BatchSize \times 16$	
Layer 1	Dense 128 + activate function ReLU	
Layer 2	Dense 256 + activate function ReLU	
Layer 3	Dense 512 + activate function ReLU	
Output layer	Dense 399 + activate function Sigmoid	

Table 18: Layers of VAE model

doi:10.6342/NTU202203944

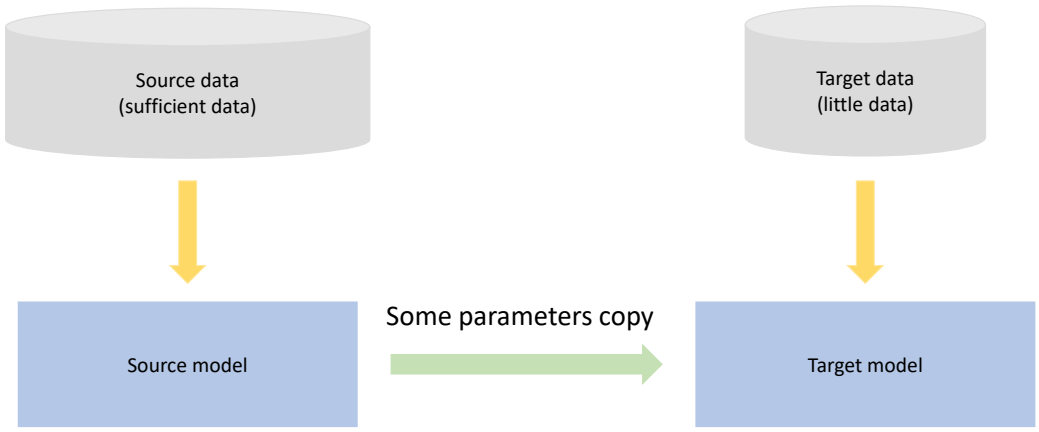


Figure 34: Illustration of model fine-tuning

slightly different distributions. This situation requires decision-level fine-tuning of the model. In the second case, the target domain is not directly related to the source domain, but the source model already has a good feature extraction capability and the target domain data has similar features to the source domain data. We consider the localization deployment in both cases mentioned above.

4.2.1.1 Layer Transfer

Among the transfer learning methods, layer transfer is a common method. The overview of the layer transfer is shown in Figure 35. Usually, if the target domain data is sufficient, we can transfer the parameters of the some layers to the target model for fine-tuning. If the target domain data is small, we can transfer the parameters of some layers to the target model and train the parameters of the rest layers that are not transferred from source model [43]. A model usually consists of two parts, the first few layers near the input layer extract the features that are useful for recognition, and the last few layers near the output layer is to classify or regress the features. In speech recognition applications, the waveforms of each person's voice may differ significantly, but the semantic meaning may be similar. When a speech recognition model is used as the source model, the parameters of the last few layers of the source model is transferred to the new model. In image recognition applications, different kinds of pictures may have similar features. In image recognition applications, the convolutional layers in the source model are transferred to the new model. In this thesis, we consider the LTE CSI vector as a

doi:10.6342/NTU202203944

radio snapshot. The patterns contained in the radio snapshot are only related to the environmental conditions, and no individualized patterns are present.

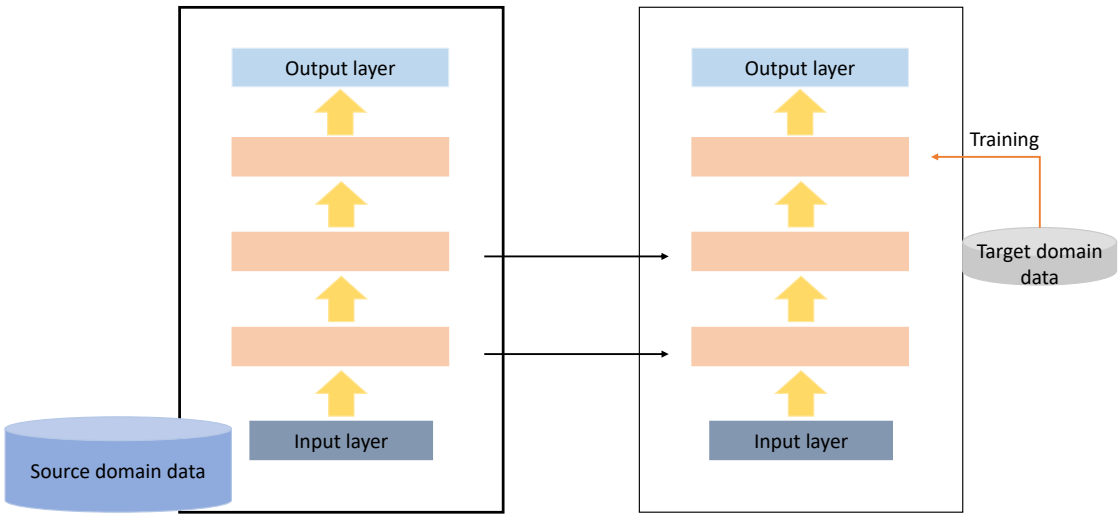


Figure 35: Illustration of layer transfer

In this thesis, we mainly try two methods of weight transfer, one is to transfer only the part of the feature extraction layers, and the other is to transfer all the layers except the decision layer to the new model. We define the input layer to the concatenate layer of MICNN as the feature extraction layers. In our experiments, we do not fix the parameters to train the model, and the transferred layers need to be fine-tuned.

4.2.1.2 Conservative Training

In machine learning applications, if there is not enough data to train a model, overfitting can easily occur when training the model. Adding some training constraints can avoid overfitting while training a model. Regularization is a method to avoid overfitting by applying regularization when training a model. The use of regularization in fine-tuning the model can also make the parameters of the fine-tuned model not differ too much from those of the source domain model. The details of adding regularization restrictions to the training model are as follows,

$$\begin{aligned} \min_{f(\cdot)} \quad & \frac{1}{N}\sum_{n=1}^{N} L(y_n, \hat{y_n}) + \lambda_1 \sum_{i=1}^{M} |w_i| + \lambda_2 \sum_{i=1}^{M} (w_i)^2 \\ \text{s.t.} \quad & \hat{y_n} = f(v_n), \end{aligned} \tag{4.2}$$

where $\frac{1}{N}\sum_{n=1}^{N} L(y_n, \hat{y_n})$ is the loss function, $\lambda_1 \sum_{i=1}^{M} |w_i|$ and $\lambda_2 \sum_{i=1}^{M} (w_i)^2$ are L1 and L2 regularization penalty, λ_1 and λ_2 are the L1 and L2 regularization factor,

doi:10.6342/NTU202203944

w_i are the model parameters of $f(\cdot)$. L1 regularization causes the less important weights in the model to shrink to 0. L2 regularization causes all the weights in the model to shrink to 0 but not to 0 as much as possible.

4.2.2 Model Generalization

The model after fine-tuning often leads to catastrophic forgetting. Catastrophic forgetting means that the model forgets much of the knowledge learned from the source domain, making the model perform worse in the source domain after fine-tuning. On the other hand, retaining the important knowledge of the source domain allows the model to have better generalizability. This method is called Lifelong learning (LLL) [44], continual Learning, and Incremental learning. LLL methods can be divided into three categories, Replay-based methods, Regularization-based methods, and parameter isolation methods. The regularization-based approach is a relatively mature method, and we will focus on basic LLL method and Elastic Weight Consolidation (EWC) [45] in this section.

The regularization-based method can be written the loss function as the following equation,

$$L_t = L_t(\theta) + \eta \sum_i b_i(\theta_i - \theta^*_{s,i})^2, \tag{4.3}$$

where $L_t(\theta)$ is the loss of the target task, $L(\theta)$ represents the loss function of the problem, $\sum_i b_i(\theta_i - \theta^*_{s,i})^2$ is the regularization term, θ_i represents the i^{th} parameter of the current model, b_i represents the value of the i^{th} parameter of the model after training on the source task, b_i represents the parameter guard of the i^{th} parameter, η denotes the importance factor of source task. When the parameter guard b_i is larger, the parameter θ_i is harder to update. The basic LLL method assumes that all parameter guards are 1, and the distance sum between θ and θ_s is very small.

Elastic weight consolidation EWC is used to calculate the parameter guards based on the fisher information matrix (FIM), but the FIM is very difficult to find. As a result, the Laplace approximation is used to obtain an approximate solution [46]. The parameter guard equation is as follows:

$$F = \left[\bigtriangledown log\left(p\left(y_s|x_s,\theta_s\right)\right) \bigtriangledown log\left(p\left(y_s|x_s,\theta_s\right)\right)^T\right], \tag{4.4}$$

where b_i is the i^{th} diagonal element in the F matrix, θ_s is the model parameters trained on source task, $p\left(y_s|x_s,\theta_s\right)$ is the posterior probability given the model parameters θ_s, source task data x_s and source task label y_s.

However, the output class of our target domain model is different from the output class of the source domain model in our system. Therefore, the parameter guards of the output layer of the source domain model cannot be applied to the

doi:10.6342/NTU202203944

target domain model, so there is no regularization restriction on the output layer of the target domain model.

Child tuning In addition to continual learning method, we also tried another fine-tuning method, which is called child-tuning [47]. When the training set is small, but the pre-trained model is large, fine-tuning may lead to worse model generalizability, and unstable model performance. Child-tuning is a method to solve the problem of unstable model fine-tuning when the training set is too small.

The paper [47] proposes two child-tuning methods, Child-tuning F and Child-tuning D, respectively. The difference between these two methods is that child-tuning randomly set gradient masks from Bernoulli distribution, while child-tuning D applied FIM to fin the gradient mask for fine-tuning. Figure 36 is the illustration of child-tuning. Some parameters are restricted by gradient masks that cannot be updated. Child-tuning F generates random gradient masks when the parameter will be updated.

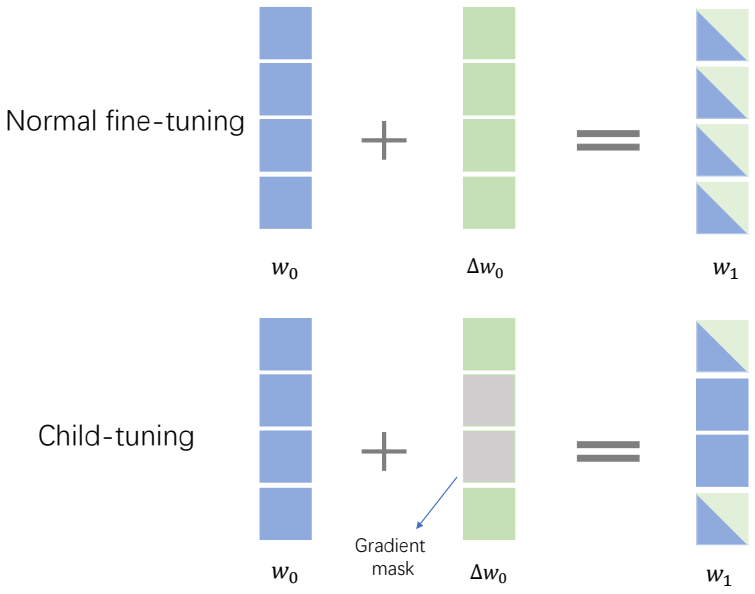


Figure 36: Illustration of Child-tuning

The parameter update can be written as the following equation,

$$\Delta\theta_0 = -\eta \frac{\partial L}{\partial \theta_0} \odot M, \tag{4.5}$$

where M is a gradient mask matrix whose values are 0 or 1 draw from Bernoulli distribution or FIM.

doi:10.6342/NTU202203944

4.3 Data Scaler

In our experiments, we use two features with different ranges of values for the subcarrier amplitude feature and the subcarrier phase difference feature. The value range of the subcarrier amplitude feature is $[0, 9999]$, while the value range of the phase difference feature range is $[-2\pi, 2\pi]$. Initially, we did not use data scaling for the phase difference feature because of the small value range of the phase difference feature, and the result is shown in Figure 37. We found that without scaling the phase difference data, the fine-tuning model easily leads to negative transfer [48]. If the scaler fits the target domain data distribution, the fine-tuned model also leads to negative transfer. In [32], the author also did not scale the data when training or fine-tuning the model, resulting in the negative transfer effect. We initially did some experiments where we used the MinMax scaler fitting on source domain data to scale the subcarrier amplitude without scaling the phase difference, resulting in the orange and red lines in Figure 2. When we scale the phase difference features and subcarrier amplitude features, the results became blue and green, and the fine-tuned model performed better than the target model. The purple line shows that we scale both features according to the target domain data distribution, and the result is that the fine-tuned model has a slight negative transfer effect.

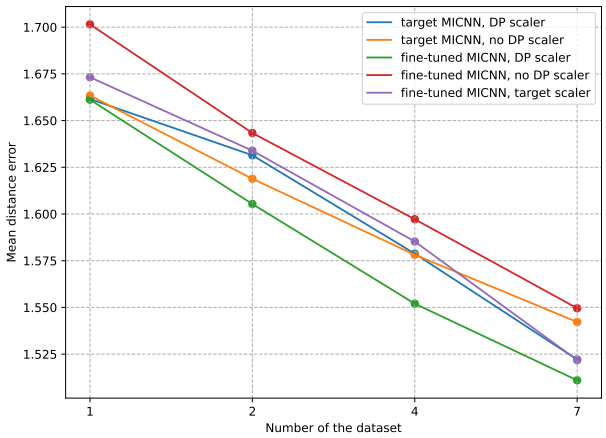


Figure 37: Comparing the MDE with or without phase difference scaler

In [49], the author use several ML algorithms with various data scalers to compare the model performance, and some ML algorithms will perform better with the suitable data scaler. Our experience is that using data scalers for features makes the models less likely to have a negative transfer effect, so we investigate the effect of different data scalers on fine-tuned model performance. In this thesis, we mainly compare the model performance of MinMax scaler (MM), Standard

doi:10.6342/NTU202203944

scaler (SS), Normalizer (NR), MaxAbs scaler (MA), Robust scaler (RS) for target model and fine-tuned model. We found that the standard scaler, MaxAbs scaler, and MinMax scaler have similar performance on the target model, and the target models perform better. We find that the range of values has more influence on the model training stability and fine-tuning model performance. Therefore, it is confirmed that it is most helpful to use MinMax scaler for model fine-tuning.

4.4 Summary

We have tried two approaches, one is to apply data augmentation to improve system performance, and the other is to use the transfer learning method to improve system performance. In data augmentation researches [42], [50], the papers used RSSI or CSI subcarrier amplitude figures as the model input feature. We apply data augmentation methods, and the CNN model performance is improved. However, the simulated phase difference of subcarriers generated by GAN or VAE cannot improve the model performance.

We discussed that the effects of different scalers and scaling ranges on the fine-tuning model are discussed in detail. The conclusion is that the best fine-tuning result can be obtained by using the MinMax scaler with the default setting.

The authors propose a fine-tuning method based on the Lifelong learning approach to fine-tune the model in [32]. However, we found that there is not much correlation between the label space of different domains, and forcing to prevent the weight update will decrease the fine-tuned model performance on the target domain. Therefore, we propose the simple fine-tuning method for model fine-tuning, which is described in Algorithm 2. The fine-tuning steps are as follows:

1.) The target domain data must be scaled by the MinMax scaler fitted on the source domain data. The scaling range is $[0, 1]$.

2.) Transfer the weights of all source model to the target model of the model except the final decision layer.

3.) The loss function plus the L2 regularization term when fine-tuning the model, and we set the regularization factor to 0.0001.

doi:10.6342/NTU202203944

Algorithm 2 Proposed model fine-tuning algorithm for indoor localization

Require: $\mathbf{T}_s$ denotes data scaler fitted on source domain data, $\mathbf{M}_s$ denotes the model trained on source domain data, $\mathbf{M}_t$ denotes the new target model, $(\mathbf{X}_t, \mathbf{Y}_t)$ denotes target training set.

Ensure: fine-tuned model $\mathbf{M}_{s\mapsto t}$

1: X_t = scaler.transform(X_t)
2: $\mathbf{M}_{s\mapsto t} = Transfer(M_s, M_t)$ % transfer all layers of $\mathbf{M}_s$ to $\mathbf{M}_t$ except final decision layer
3: $\mathbf{M}_{s\mapsto t}$.train(X_t,Y_t)
4: return $\mathbf{M}_{s\mapsto t}$

doi:10.6342/NTU202203944

CHAPTER 5

PERFORMANCE EVALUATION

5.1 Datasets

We mainly collected and established our datasets in Rooms 521 and Rooms 114 of the National Taiwan University Barry Lam Hall. The two rooms are called BL521 and BL114 respectively. The data we collected in Room 521, called the domain 1 dataset, contains the data of eight paths. The data set collected in Room 114, called the domain 2 dataset, contains the data of four paths. We obtain the relative coordinates from the SLAM algorithm. To convert the relative coordinates to our actual field coordinates, we just rotate the relative coordinates and add the bias (initial field coordinates).

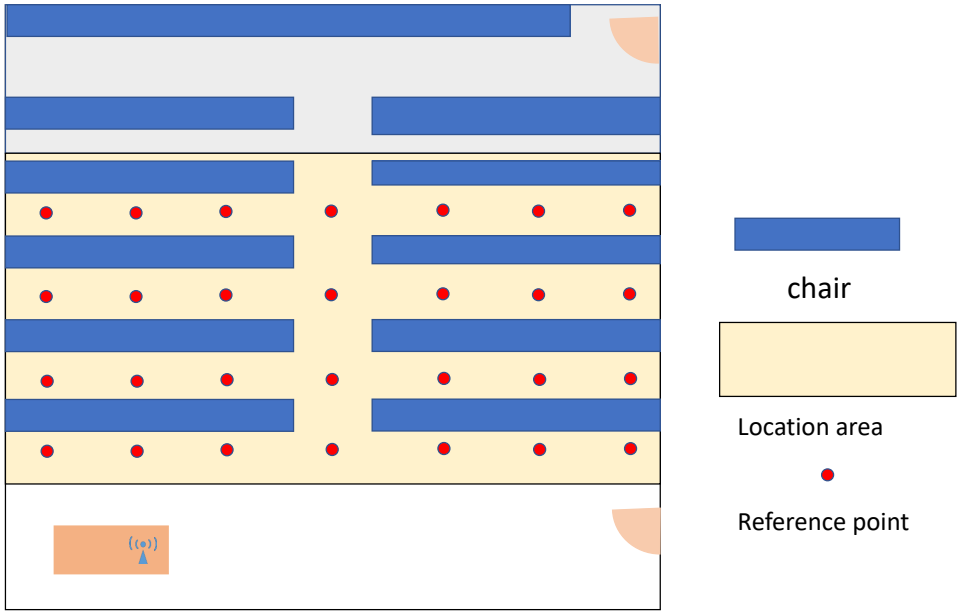
5.1.1 Domain 1 Dataset

We collected the domain 1 dataset in BL521. The floor plan of BL521 is shown in Figure 23. We set 12 reference points and map the dataset coordinate labels to the closest reference point. The data quantity of each trajectory dataset is shown in Table 10. The domain 1 dataset contains eight trajectory datasets, and each trajectory dataset includes 3500-5800 labeled data. The trajectories of data collection are shown in Figures 24. We designed the path to pass through all reference points (cluster points), and the method can help train and evaluate the model.

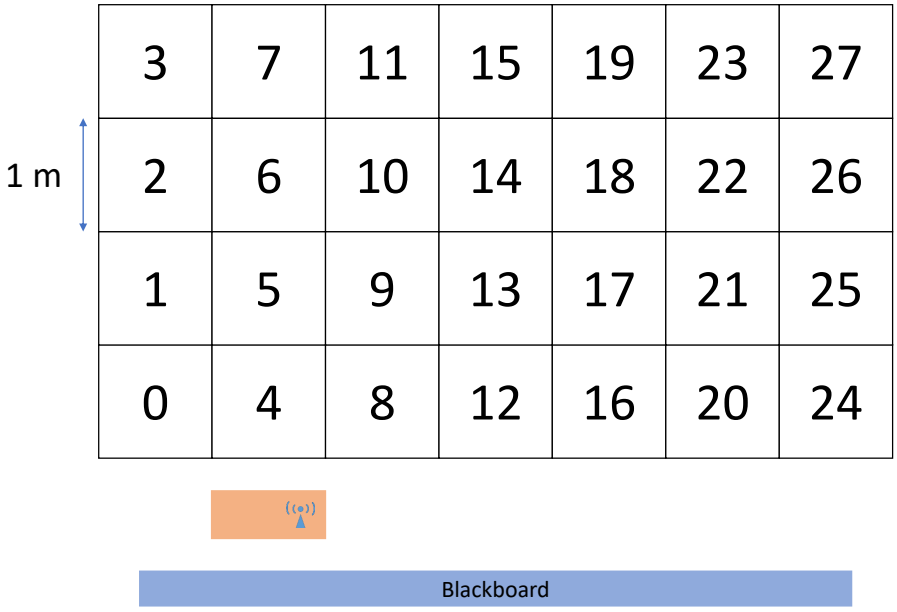
5.1.2 Domain 2 Dataset

The plan view of BL114 is shown in Figure 38(a). There is a step between the light yellow and light blue of BL114. The robot cannot climb up the step by itself, and the back area of the classroom is covered in the positioning area. The base station is set up on the lecture Table. The light yellow color is the positioning area. We set 28 reference points in BL114 and gave each reference point a number for easy analysis, as shown in Section 38(b). The trajectories of the domain two datasets are shown in Figure 2. The data amount of domain two datasets are shown in Table 19. The data amount of features in each trajectory dataset is about 11,000-22,000.

doi:10.6342/NTU202203944



(a) Floor plan of BL114



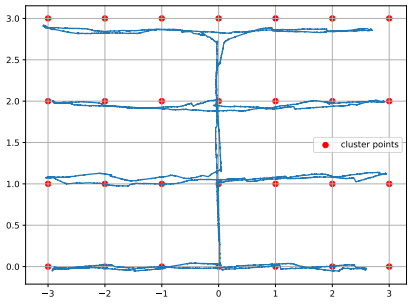
(b) The reference points numbers

Figure 38: The BL114 localization system deployment

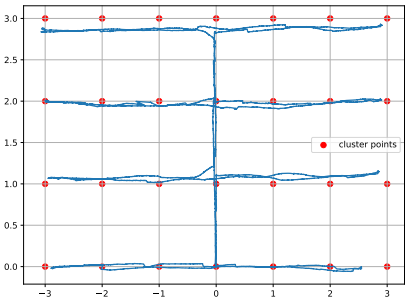
doi:10.6342/NTU202203944

data quantity	
Trajectory 1	16952
Trajectory 2	19990
Trajectory 3	19539
Trajectory 4	21707
Trajectory 5	14490
Trajectory 6	17079
Trajectory 7	12197
Trajectory 8	15064
Trajectory 9	11161
Trajectory 10	11046

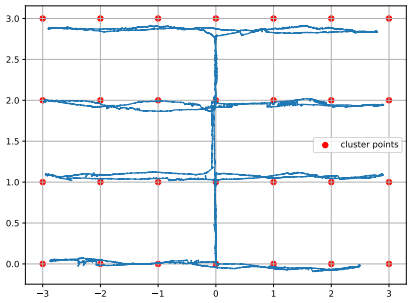
Table 19: The data quantity of domain 2 dataset



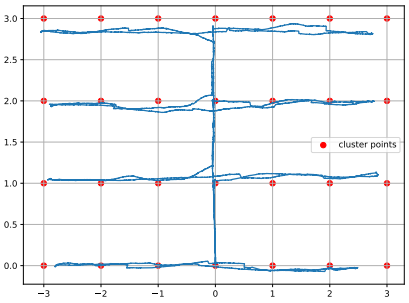
(a) Trajectory 1



(b) Trajectory 2

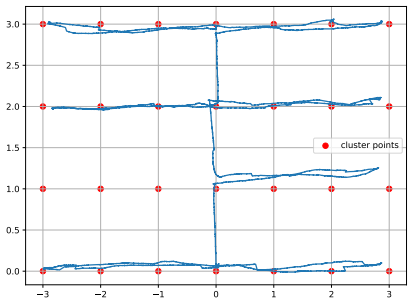


(c) Trajectory 3

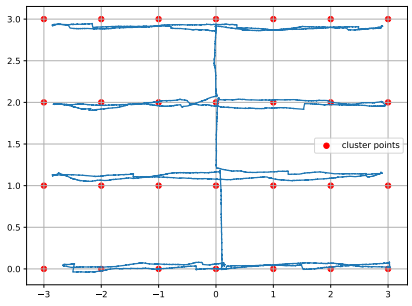


(d) Trajectory 4

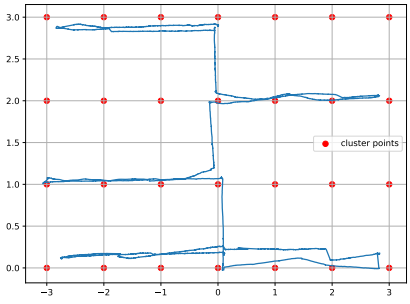
doi:10.6342/NTU202203944



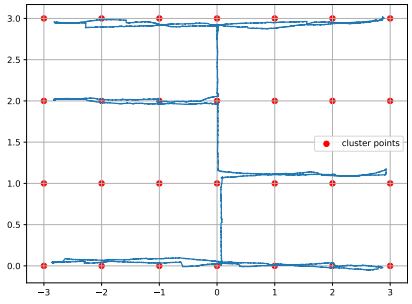
(e) Trajectory 5



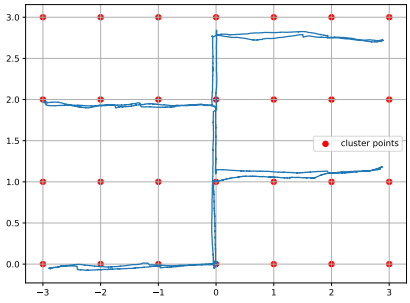
(f) Trajectory 6



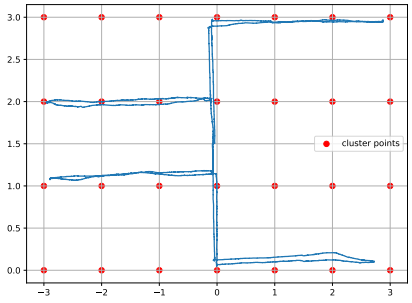
(g) Trajectory 7



(h) Trajectory 8



(i) Trajectory 9



(j) Trajectory 10

Figure 39: The trajectories of domain 2 data

doi:10.6342/NTU202203944

data quantity	
Trajectory 1	6545
Trajectory 2	6655
Trajectory 3	6603
Trajectory 4	7332
Trajectory 5	6889
Trajectory 6	6787
Trajectory 7	6780
Trajectory 8	6369

Table 20: The data quantity of domain 3 dataset

5.1.3 Domain 3 Dataset

A $1.8m \times 9m$ corridor area on the fifth floor of Barry Lam Hall was selected as a positioning area. The floor plan of the collection area is Figure 40. We selected 20 points as reference points in this area, and the distance between the adjacent reference points is 0.9 m. We collect a total of eight trajectory datasets, and Figures 41 show the eight trajectories estimated by SLAM algorithm and manually corrected. Each trajectory dataset contains between 6300-7400 fingerprints, and the exact data amount of trajectory datasets are shown in Table 20.

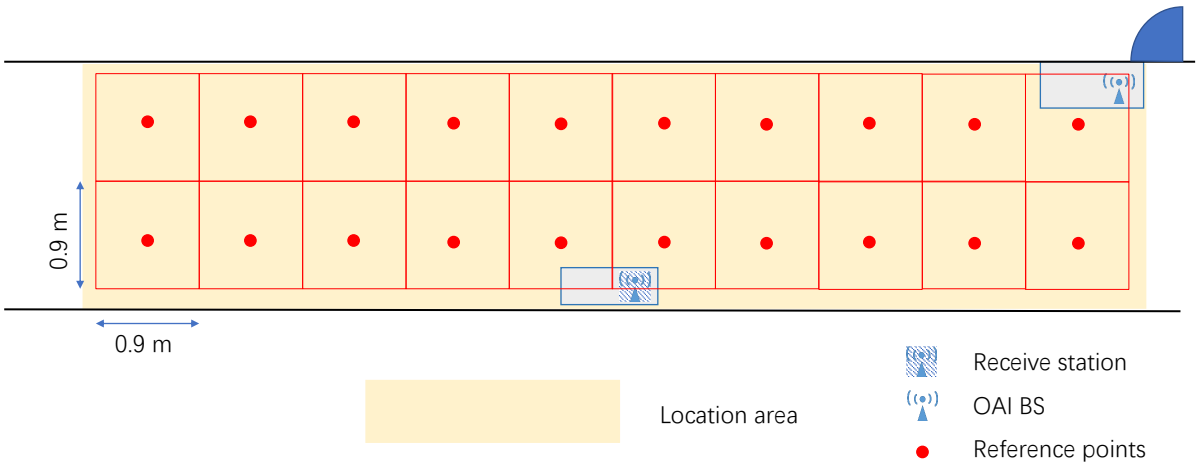
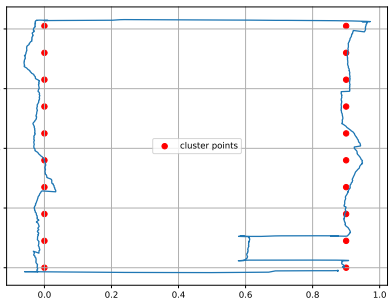
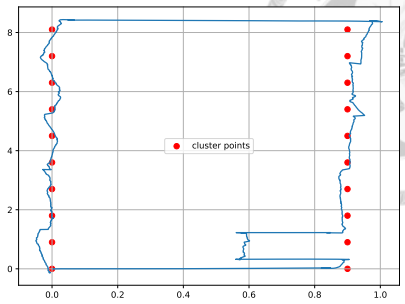


Figure 40: Floor plan of BL5F corridor

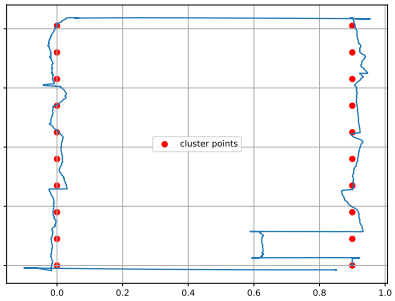
doi:10.6342/NTU202203944



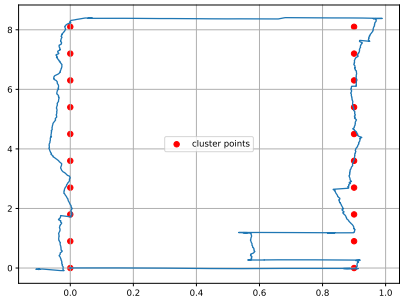
(a) Trajectory 1



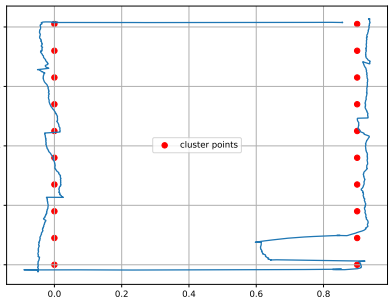
(b) Trajectory 2



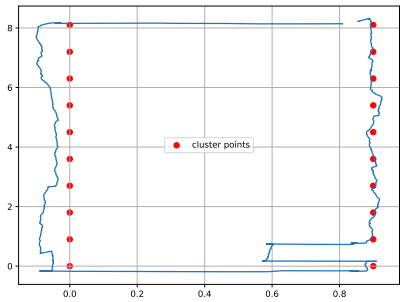
(c) Trajectory 3



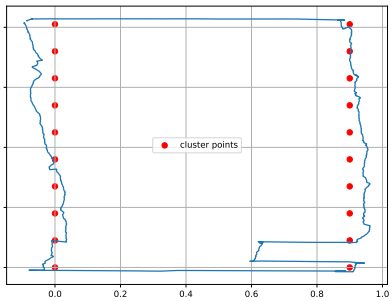
(d) Trajectory 4



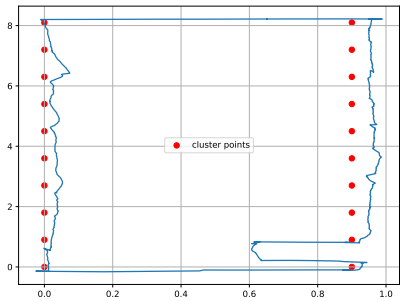
(e) Trajectory 5



(f) Trajectory 6



(g) Trajectory 7



(h) Trajectory 8

Figure 41: The trajectories of domain 3 data

doi:10.6342/NTU202203944

5.2 Evaluation of Data augmentation

We used the data collected in the corridor outside of BL521 (domain 3 dataset) to experiment with the feasibility of generating the model.

5.2.1 Considering Data Augmentation with GAN

First, we use one and seven trajectory data as the training set to evaluate the feasibility of GAN. Figure 42 shows the result for the training set containing one trajectory data. There are 20 reference points in domain 3, and GAN generates 100 fake data for each reference point. We try to use different amounts of dummy data (500, 1000, 1500, 2000) to merge with the training dataset for model training. From the results in Figure 42, the fake data of amplitude is helpful to improve the model performance, but the fake data of phase differences confuse the model judgment. From the comparison of the two T-SNE visualizations in Figures 43, we believe that GAN has mode collapse problem for generating phase difference features.

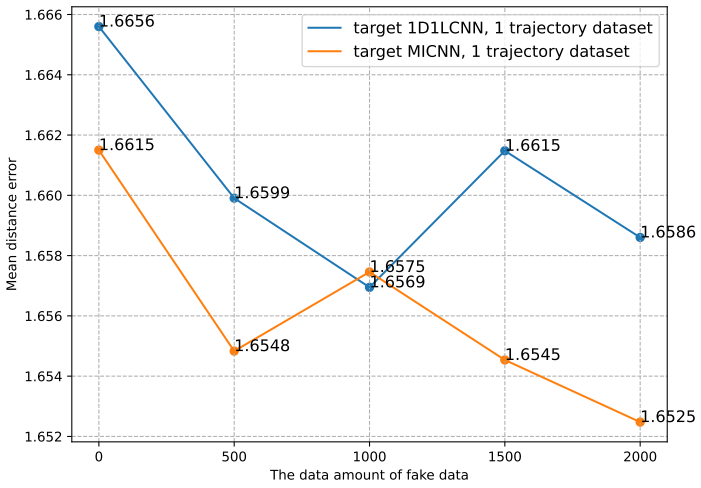
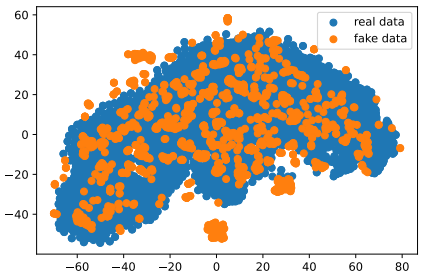


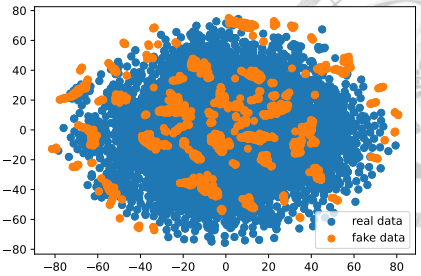
Figure 42: Comparison the MDE with fake data, 1 trajectory dataset

We use seven trajectory datasets as the training set for GAN. Figure 44 shows the effect of using different amounts of fake data on the model performance. We found that using seven trajectory datasets to train the GAN does not contribute to the model performance. Except for the blue line, which has a slight improvement, the model performances with different conditions are relatively poor with fake data. Our data analysis in Figure 45 contrasts that GAN tends to ignore a lot of relatively small-scale distributions. From the two t-SNE analyses in Figure 45, the fake data distribution is more concentrated and the fake data generated by GAN

doi:10.6342/NTU202203944



(a) Feature visualization of subcarrier amplitude using t-SNE.



(b) Feature visualization of subcarrier phase difference using t-SNE.

Figure 43: Simulated data analysis

cannot be similar to all the real data. This situation is somewhat like a slight mode collapse of the GAN. GAN has severe mode collapse problem for generating subcarrier phase difference.

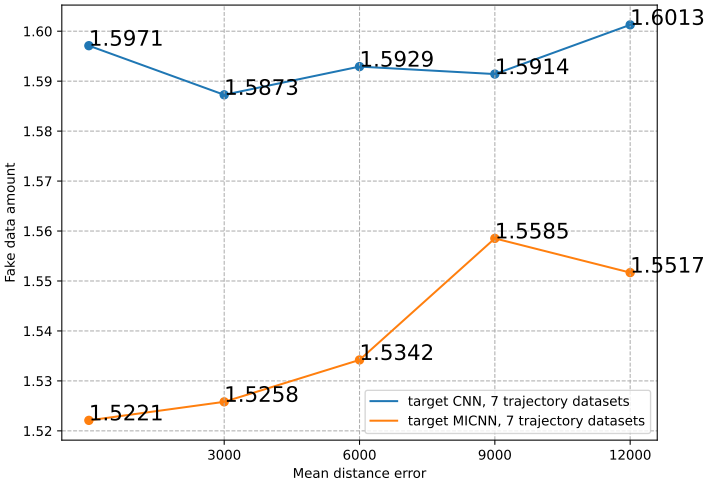
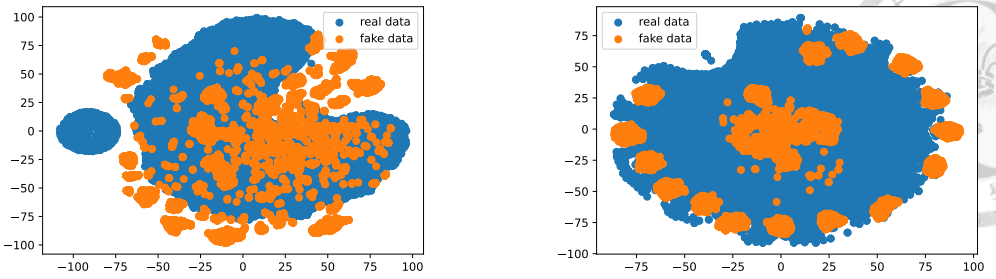


Figure 44: Comparison the MDE with different amount of fake data, 7 trajectory datasets

5.2.2 Considering Data Augmentation with VAE

We use VAE to generate dummy data for subcarrier amplitude and subcarrier phase difference to train the model. Figures 46 show the results. We found that a large amount of training data is required to train the VAE that can generate simulated features that contribute to the performance of the CNN model. From the MICNN model result, it is observed that the fake data generated by VAE does not improve the model performance. Combining with the data analysis in Figures 47, we think the secondary features in the phase difference may be very helpful for

doi:10.6342/NTU202203944



(a) Feature visualization of subcarrier amplitude using t-SNE. Training set is (0, 1, 2, 3, 4, 5, 6).

(b) Feature visualization of subcarrier phase difference using t-SNE. Training set is (0, 1, 2, 3, 4, 5, 6).

Figure 45: Simulated data analysis

model performance, but the VAE model does not learn these secondary features very well, and the phase difference of the real data is very similar, resulting in the loss of secondary features.

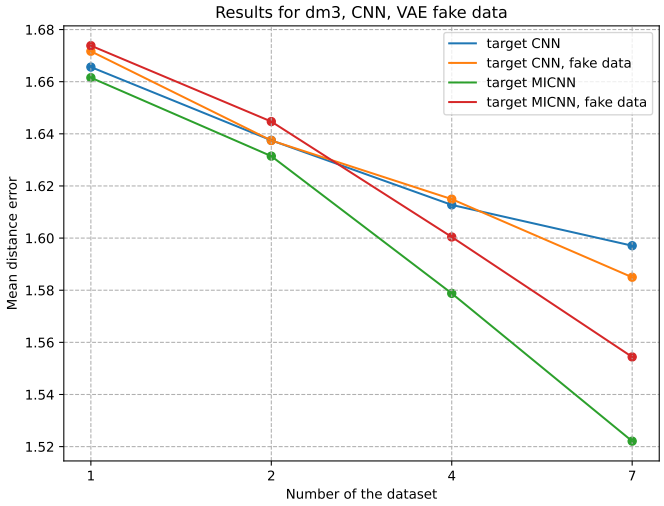
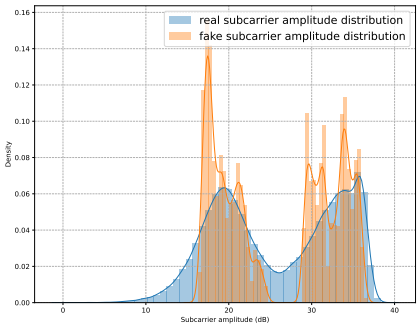


Figure 46: Comparing the MDE with fake data generated by VAE

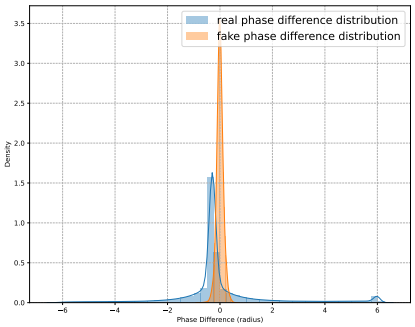
5.3 Evaluation of Transfer Learning

In our thesis, we consider learning-based indoor localization methods. We adopt an alternative cross-validation approach in our experiments to evaluate our experimental results, as described in Section 3.4.2. Our metric for validating the experimental results is described in Section 3.4.1.

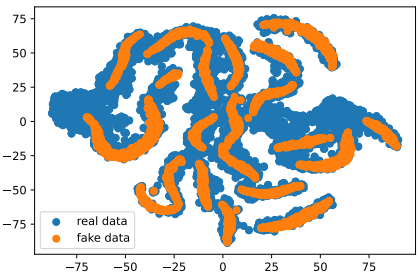
doi:10.6342/NTU202203944



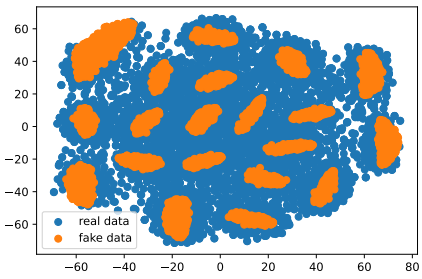
(a) Comparing the subcarrier amplitude distribution of real data and simulated data generated by VAE.



(b) Comparing the phase difference distribution of real data and simulated data generated by VAE.



(c) Feature visualization of subcarrier amplitude using t-SNE.



(d) Feature visualization of subcarrier amplitude and phase difference using t-SNE.

Figure 47: Analysis of simulated data generated by VAE

doi:10.6342/NTU202203944

5.3.1 Domain 1 and Domain 2 Case

In this experiment, we use the data of domain 1 as source domain dataset. We select a few trajectories datasets from domain 2 data as training data and the rest of the trajectories datasets as test data. For example, we have 10 data sets in domain 2, we randomly select some combinations from the C_n^{10} data sets combinations for the experiment.

5.3.1.1 Fundamental Experiment

The loss curve for different conditions is shown in Figure 48. We use all two combinations of domain 2 trajectories dataset as training dataset and complementary set of combinations as test dataset. We used the BL521 dataset as pre-training data and the BL114 dataset as target data. We re-sample 200 samples per reference point from one trajectory dataset in BL114 as the re-sampling method. We evaluate the performances of different models using the domain dataset. We use SLN and CNN models mentioned in Chapter 3. We use the same model training setup as in Chapter 3.

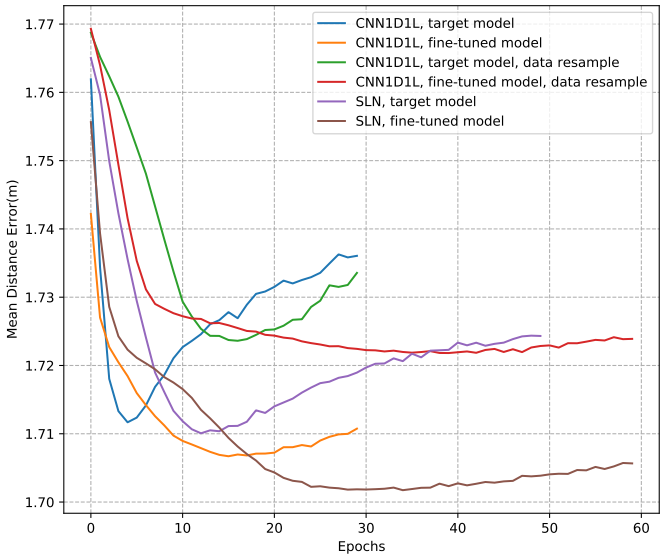


Figure 48: Loss curves of the testing data under different conditions

In Figure 48, the coffee, orange, and red curves are the loss curves using the BL521 dataset as pre-training data. Using the parameters of the pre-trained model as the parameters of the initial model makes the model less prone to overfit. No matter whether using the whole trajectory dataset or re-sampling dataset, the model performance will not vary greatly. Comparing the CNN model with the

doi:10.6342/NTU202203944

SLN model, the CNN model should be easier to learn the features and overfit the training data than the SLN model. The performance comparison of the different conditions is shown in Figures 49. The mean distance error of the pre-trained SLN model is minimized. We observe that SLN slightly fits training data compared to CNN model. It is not that the feature extraction ability of the CNN model is worse. When the amount of data is small, a simple structured model can obtain better performance.

We compared the results of fine-tuning or training under these conditions. The results are shown in Figure 49. We used two trajectories datasets to train or fine-tune the model, and the remaining eight trajectories datasets were used as test data. We know from the results that the fine-tuned model is slightly better than the newly trained model.

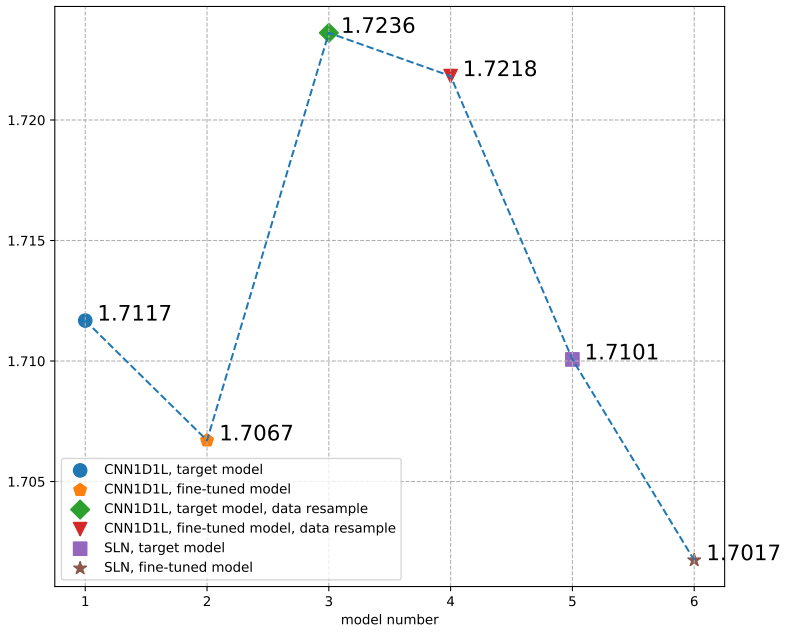


Figure 49: Comparing the MDE of the models under the different conditions

5.3.1.2 Considering Conservative Learning

In our previous experiments, we found that the accuracy of the model trained on two trajectories datasets was not accurate, and we started to consider picking any 20 training dataset combinations from the C_5^{10} training dataset combinations training set combination of five for performance evaluation,and this allows each training set to contain five trajectories datasets. The result is shown as below Figure 50. We investigate the effectiveness of conservative learning. We compare

doi:10.6342/NTU202203944

the performance of CNN with L1 and L2 regularizers. The CNN model with 10^{-6} L1 regularization factor is slightly better than the CNN model with 10^{-6} L2 regularization factor. Increasing the L1 regularization factor to 10^{-4} makes the model performance worse. The model performance with L1 and L2 regularizers are similar to the model performance with L1 regularizer. The model performance with 10^{-4} L2 regularization factor is slightly better than the model performance with 10^{-6} L2 regularization factor. We also evaluate the performance of the SLN

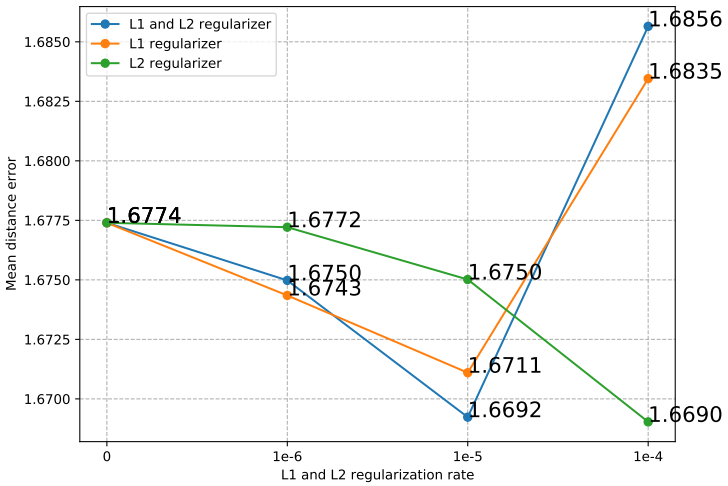


Figure 50: Comparing the MDE of the CNN model with regularization

model with regularizers, and the result is Figure 51. It is observed from Figure that SLN with L2 regularizer has almost no effect, and the most improvement is in SLN with L1 and L2 regularizers. The improvement trend is similar to that of CNN, but the performance of CNN with regularizers has slightly better results.

5.3.1.3 Considering Layer Transfer

Here we discuss the effectiveness of the layer transfer for our application. This method has been discussed in Section 4.2.1.1, and the model framework has been presented in Section 3. Figure 52 is the result. The blue line is the result of fixing the parameters transferred from the source model without fine-tuning. The orange line is the result of fine-tuning the whole model after transferring the parameters.

From the results, the feature extraction layers of the source model are relocated to the new model to make the fine-tuned model fall into slightly better regional minima. The decision layers of the source model are not helpful in determining the class of the target domain.

doi:10.6342/NTU202203944

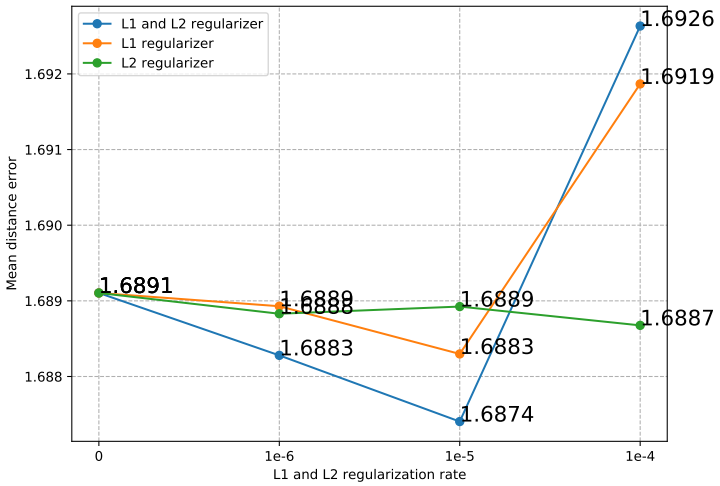


Figure 51: Comparing the MDE of the SLN model with regularization.

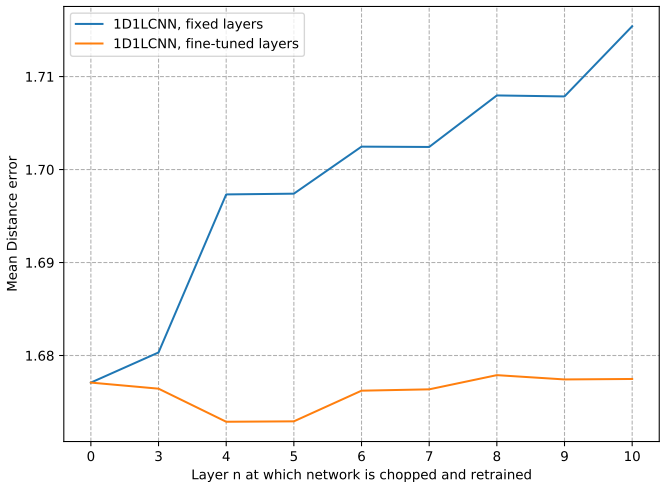


Figure 52: Comparing the MDE of the model with different layers transfer

doi:10.6342/NTU202203944

5.3.1.4 *Discussing the Impact of the Data Amount of Target Domain Training Data on the Model*

From the previous results, we observe that the two trajectories datasets are relatively difficult to obtain a good performance. We compare the models performances with the increasing amount of data. We use 2,5,7,9 trajectories datasets as training data and the rest of the datasets as test data. We observe in Figure 2 that the fine-tuned pre-trained model does obtain better results when the training data is small. As the data increases, training a new model achieves better performance than the performance of the fine-tuning model. However, this result compares poorly with the results we obtained in Chapter 3. The performance of the models is only improved by about 6 cm when using two trajectories datasets as training data compared to nine trajectories datasets as training data.

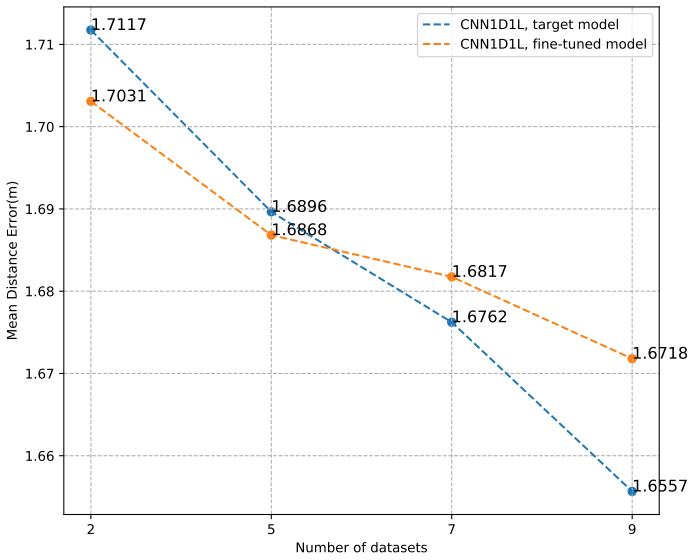


Figure 53: Comparing target CNN and fine-tuned CNN

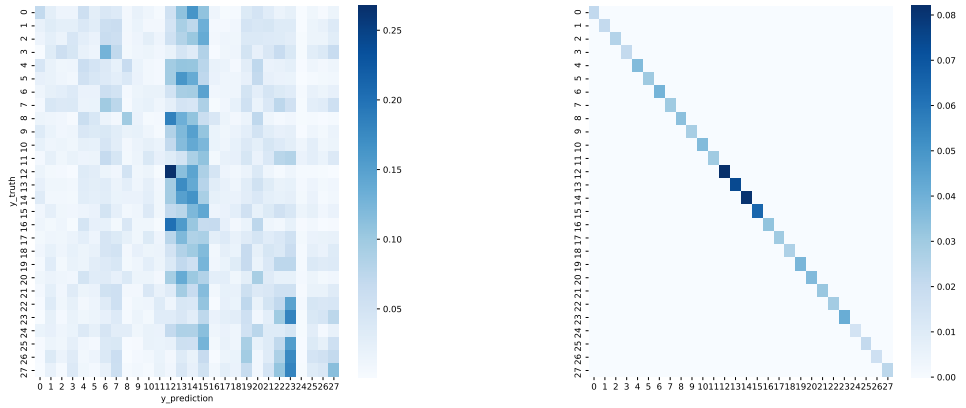
5.3.1.5 *Model Output Analysis and Feature Analysis*

Figure 54(a) shows an average confusion matrix for model output. We obtain the confusion matrix for various combinations of training sets, and we average multiple confusion matix to obtain the results in Figure 54(a). The numbers of the positions are shown in Figure 38(b). The numbers 12-15 are the area in the middle aisle of the classroom. Figure 54(b) is the average label distribution of the trajectories datasets.

We observe from the results that the accuracy of the model is poor, but

doi:10.6342/NTU202203944

there is still some regularity in the predicted results of the model. In addition, the labels 0,4,8,12,16,20,24 are more likely to be confused by the model. The labels 3,7,11,15,19,23,27 are also easily confused or incorrectly identified as 21,22,23,25,26,27. Overall, the accuracy of the model is not good enough but incorrectly identified as other labels are still some rules. We think about several possibilities, 1.) We know from Figure 54(b) that the labels in the data are not evenly distributed, and the uneven data may cause the model output to be easily misidentified. 2.) The fingerprints collected in different locations easily cause confusion, and we should think about how to increase the feature recognition of the data.



(a) Average model output distribution

(b) Average label distribution of domain 2 datasets

Figure 54: Model output and label distribution confusion matrix

5.3.1.6 Considering Transfer Learning with Data Balancing Algorithms for Domain 2

We speculated in the previous analytical experiments that the poor performance of the model may be due to imblanced data. Therefore, we used several methods to balance the data labels, namely SMOTE [51], RandomOverSampling, and RandomUnderSampling. RandomOverSampling and SMOTE add fake data or repeated data to the training set. We compared the results with and without the data balancing algorithm in Figure 2. We found that no matter which data sampling strategy is used, the results are no better than those without using any sampling strategy.

doi:10.6342/NTU202203944

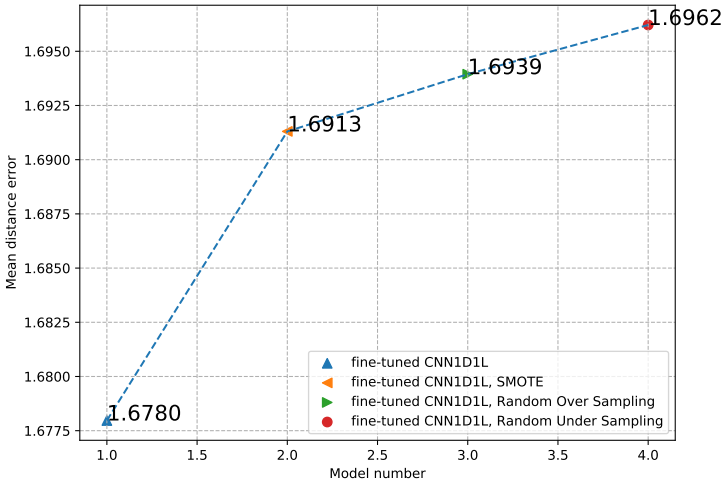
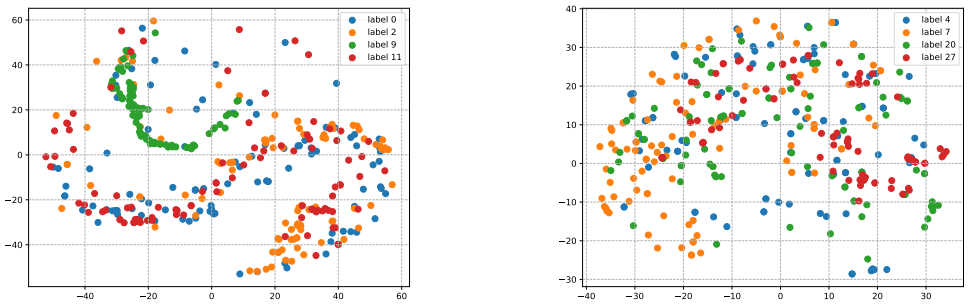


Figure 55: Comparing with or without different Random Sampling methods

5.3.1.7 Using t-SNE Data Visualization for Domain 1 Data and Domain 2 Data

We use the t-SNE [52] to analyze our data. We concatenate all the data of the domain for visualization and then pick out the four corners of the map from the transformed data to plot. We visualize the domain 1 data and domain 2 data, and the results are shown in Figure 56. The data of domain 2 with the same label are less aggregated. From the two results, we observe that the difference between the data of domain 1 is greater than that of domain 2. Therefore, it is expected that the performance of domain 1 has better accuracy than the performance of domain 2.



(a) Select the domain 1 LTE CSI vectors of the specific area $0, 2, 9, 11$ then T-SNE

(b) Select the domain 2 LTE CSI vectors of the specific area $4, 7, 20, 27$ then T-SNE

Figure 56: Data visualization for domain 1 and domain 2

We additionally make the confusion matrix of the target model of domain 1, and we observed from Figure 57 that the localization performance in domain 1

doi:10.6342/NTU202203944

is better than localization performance in domain 2, possibly due to the factors in the environment of domain 2 that lead to poorer localization accuracy. When we collected the data of domain 1, we placed the base station on the top of the cabinet to increase the transmission distance and the possibility of direct signal transmission.

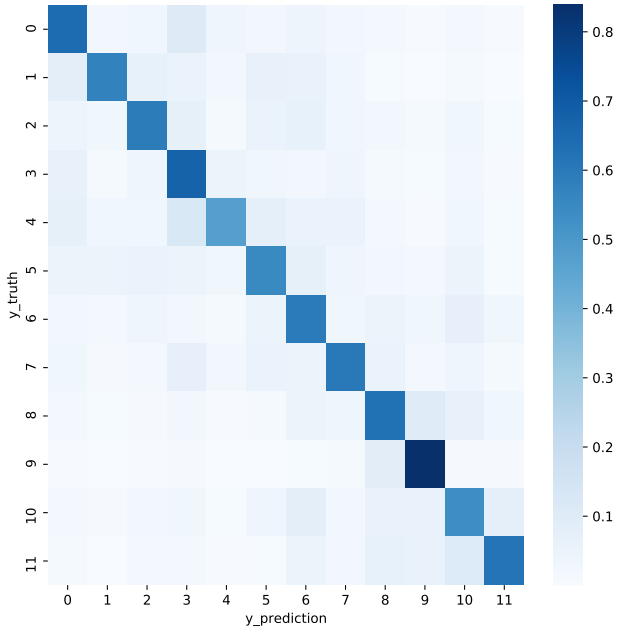
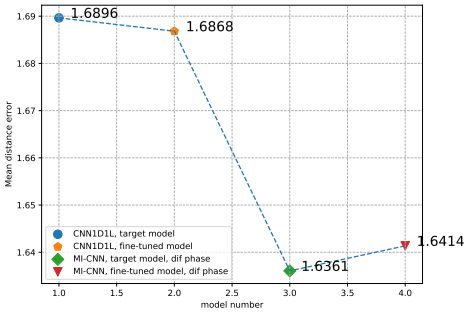


Figure 57: Confusion matrix of domain 1 model prediction

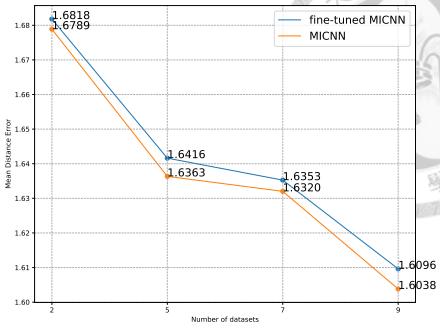
5.3.1.8 Consider Adding Phase Differences to Features and Fine-tuning the Model

We combine the old feature with the phase difference into a new model input. Since the real part of the subcarriers constitutes a feature with low correlation with the phase difference feature, we design a new model detailed in Section 3.2.5. We compare the effect of having the phase difference as a feature on the performance of the models. The model in Figure 58(a) uses several combinations of five trajectory data to train the model. Figure 58(a) shows the average MDE for the 20 combinations. We observe that adding phase difference as a feature can improve the model performance by about 3%. However, we observe from Figure 58(b) that pre-training with source domain data has a negative transfer effect on the target domain. The negative transfer [53] may be due to the large difference in the phase difference distribution between the source domain and the target domain.

doi:10.6342/NTU202203944



(a) Comparing with or without phase difference based on target model and fine-tuned model. The models are trained on 5 trajectory datasets.



(b) Comparing with different amount of trajectory datasets based on target MICNN and fine-tuned MICNN

Figure 58: Comparing the model performance of adding phase difference

5.3.1.9 Result Summary Based on Domain 2

We took BL114 as the target domain and BL521 as the source domain to experiment. We found that there was less useful knowledge that could be transferred between the two domain feature spaces, probably due to topography, area, and environment. Therefore, we collected data in the corridor outside of BL521 as target domain data would be discussed in detail in the next subsection.

5.3.2 Domain 1 and Domain 3 case

5.3.2.1 Consider Model Fine-tune Based on Domain 3 Dataset

We compare the average MDE of MI-CNN model with three fine-tuned methods. Fig. 59 and Table 21 show the results. The "fine-tuned" model represents the initial model by taking the weights of all layers of the source domain model except the decision layer to the new model. The model regularization method is described in Chapter 4, and we try to use the conservative training (L2 Reg.) based on the previous results, and L2 factor is 0.0001. "FEL transfer" means that we transfer the feature extraction layer from the source domain model to the new model as the initial model. "different LR" means that we use two learning rate to fine-tune model, and we set the learning rate of feature extraction layers to be one tenth of the learning rate of other layers.

From the results of the MICNN model, it is observed that the fine-tuning model performance is better than the performance of the target model. When the training set contains one trajectory dataset, the training set has a larger bias resulting in the performance of the fine-tuned model being similar to the performance of the target model. When the training set contains seven trajectory

doi:10.6342/NTU202203944

conditions	Number of domain 3 trajectory dataset			
Model	MDE (1)	MDE (2)	MDE (4)	MDE (7)
target MICNN	1.6615	1.6314	1.5788	1.5221
fine-tuned MICNN	1.6570	1.6059	1.5521	1.5093
fine-tuned MICNN, L2 Reg.	1.6582	1.6012	1.5519	1.5068
fine-tuned MICNN, FEL transfer	1.6515	1.6074	1.5607	1.5191
fine-tuned MICNN, Dif LR.	1.7151	1.6175	1.5692	1.5303

Table 21: Comparison of different model with different conditions

datasets, the performance of the fine-tuned model is not much better than the performance of the target model.There are two possible reasons for this situation: 1) The source domain space is smaller than the target domain space, so little knowledge of the source domain can be transferred to the target domain. 2) The source domain dataset contains a smaller data amount of fingerprints than the target training set, resulting in less information being transferred.

We compare the four fine-tuning methods, and the performance of the four fine-tuning methods is similar. The best method in these methods are fine-tuned MICNN and fine-tuned MICNN with conservative training (L2 Reg.), when training set contains two or 4 trajectory datasets. When there are seven trajectory datasets in the training set, the model performance is more variable, which is related to the fact that we have less test data.

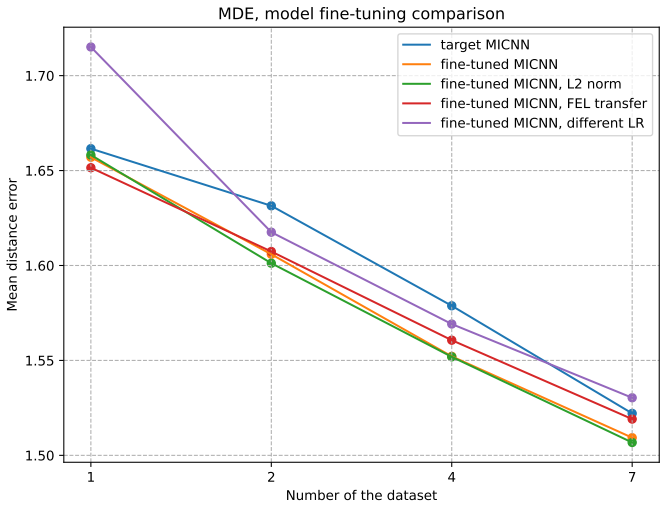


Figure 59: Comparing the performance with target model and fine-tuned model

doi:10.6342/NTU202203944

5.3.2.2 Consider Model Generalization

We applied the Lifelong learning approach to fine-tune model. The details of the approach are described in Section 4.2.2. Figure 60 shows the results of using different Lifelong learning methods. "LLL 0.01" means that the basic Lifelong learning method with lambda = 0.01 is applied to fine-tune model. "EWC 500" means that EWC with lambda = 500 is applied to fine-tune model.

From the results, we can observe that EWC is more powerful in blocking important weight updates. However, blocking the update of significant weights does not help to improve the model performance on the target domain. We think that this may be due to the low correlation of the label space between the two domains. If the label spaces of two domains are similar, using the LLL approach can increase the generalizability of the fine-tuned model. However, the label space correlation between the two domains is very low, and using the LLL approach prevents the model fine-tuning.

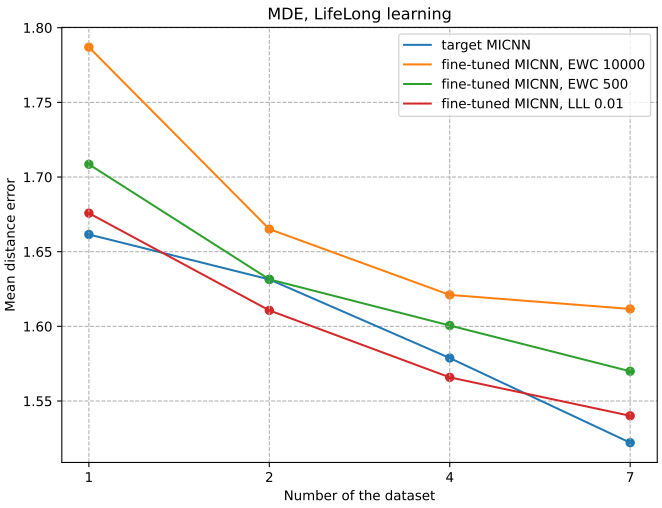


Figure 60: Comparing the average MDE for continual learning

Incremental learning can not improve the performance of the fine-tuned model in the target domain, and we think about using child-tuning to improve the performance of the model. The details of child-tuning are described in section 4.2.2. We compare the average MDE of the fine-tuned model with Child-tuningF and Child-tuningD for fine-tuning. The child-tuningD method is slightly better than the child-tuningF method when the training set contains 2 or 4 trajectory datasets.

From the results in Figure 60 and Figure 61, it is observed that the correlation between the label spaces of the two domains is very low in indoor localization applications, which is not suitable for the Lifelong learning approach. The Child-tuning

doi:10.6342/NTU202203944

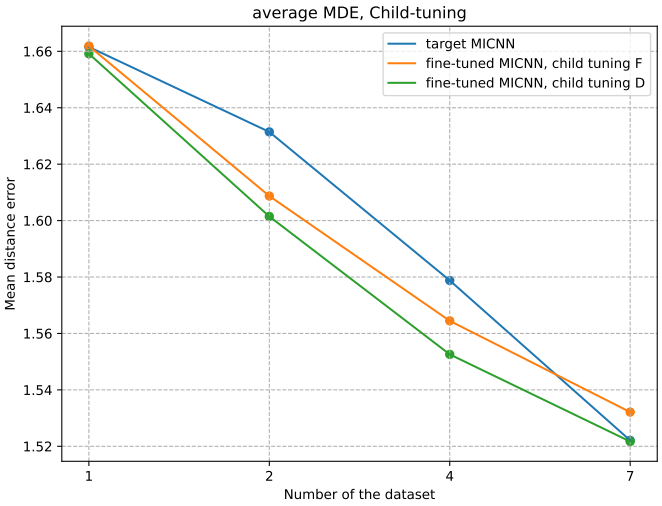


Figure 61: Comparing the average MDE for Child-tuning

method is more suitable for environmental adaptation in the indoor localization application.

5.4 Evaluation of Data Scaler

We use the MICNN model to evaluate the effect of the different scalers on the model performance. We use domain 1 and domain 3 as source domain and target domain in this section respectively.

5.4.1 Consider the Different Scaler

Comparing the performance of MICNN with different scalers, the result is Table 22. We find that the target model with Standard scaler, MaxAbs scaler, and MinMax scaler have similar model performance, but only MinMax scaler can be used to fine-tune the method without leading to a negative transfer effect.

1.) Target model performance: $SS > MA > MM > RS > NR$

2.) Fine-tuned model performance: $MM > SS > MA > RS > NR$

3.) Non-negative transfer learning: MM, NR

The distributions are shown in Figure 62. We compare the distribution of scaled target domain data." Fine-tuned" means that we use the scaler fitted on source domain data to scale the target domain data. "target" means we use the scaler fitted on target domain data to scale the target domain data. Table 23 is the value range of scaled data. We call scaler fitted on source domain as source scaler and scaler fitted on target domain data as target scaler. In our experiments, the two scalers that do not lead to a negative transfer effect on the fine-tuned model are MM and NR. We think that the occurrence of negative transfer is related to

doi:10.6342/NTU202203944

conditions	Number of domain 3 trajectory dataset				avg MDE
Model	MDE(1)	MDE(2)	MDE(3)	MDE(4)	MDE(avg)
target, MM	1.6643	1.6339	1.5795	1.5221	1.6002
fine-tuned, MM	1.6556	1.6043	1.5579	1.5020	**1.5780**
target, SS	1.6573	1.6093	1.5755	1.5495	1.5987
fine-tuned, SS	1.6765	1.6373	1.5959	1.5480	1.6112
target, NR	2.1736	1.9654	1.8930	1.8370	1.9495
fine-tuned, NR	1.9498	1.9040	1.8467	1.8087	1.8815
target, MA	1.6615	1.6235	1.5730	1.5379	1.5992
fine-tuned, MA	1.6975	1.6342	1.5851	1.5739	1.6227
target, RS	1.6840	1.6519	1.6162	1.5810	1.6333
fine-tuned, RS	1.7128	1.6644	1.6087	1.5817	1.6419

Table 22: Comparison of the model with different scaler

	source (Amp.)		target (Amp.)		source (Ph. Dif.)		target (Ph. Dif.)	
Condition	Min	Max	Min	Max	Min	Max	Min	Max
MM	**-0.004**	**1.511**	**0**	**1**	**-0.014**	**1.006**	**0**	**1**
SS	-2.037	6.279	-1.885	17.634	-5.885	5.938	-5.428	5.472
NR	0	0.247	0	0.247	-0.376	0.438	-0.376	0.438
MA	0	1.510	0	1	-1.010	1.007	-1	1
RS	-1.339	4.051	-1.241	19.515	-91.481	98.830	-91.457	101.136

Table 23: Comparing the value ranges of the scaled data

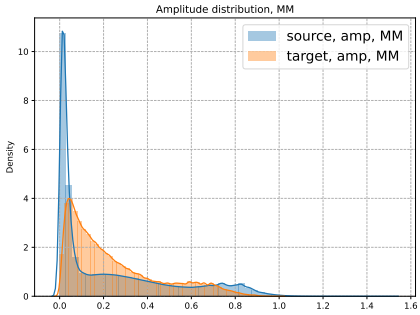
the range of values after scaling. The phase difference range of the target data scaled by MA is twice as large as the phase difference range of the target data scaled by MM, which is similar to the effect of negative transfer effect caused by the phase difference without scaling. In addition, the fine-tuning model of RS performs better when the training set has four or seven trajectory data sets, but that is the result of unstable model training.

5.4.2 Consider Scaling Range

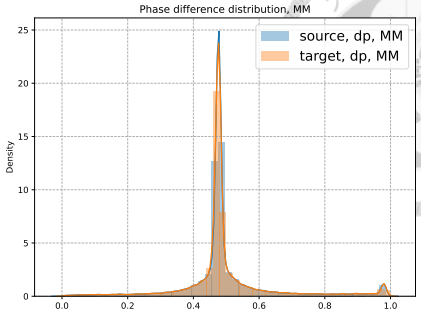
In the previous experiments, the performance of the model using the MaxAbs and MinMax scaler is relatively different, and the model does not have a negative transfer effect using the MinMax scaler. However, the model using MaxAbs scaler leads to a negative transfer effect, which we are more confused about. Therefore, we use the MinMax scaler to compare the MDE with the different scaling ranges of the MinMax scaler for fine-tuning the model.

We compare the model results for the phase difference scalers, and the scaling

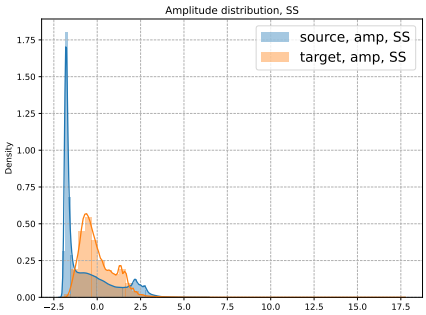
doi:10.6342/NTU202203944



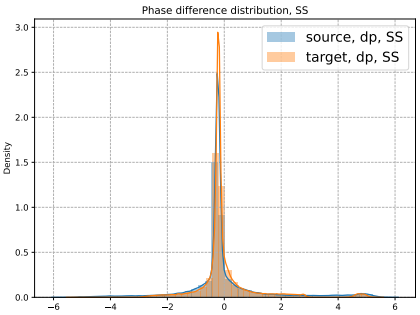
(a) Amplitude, MM



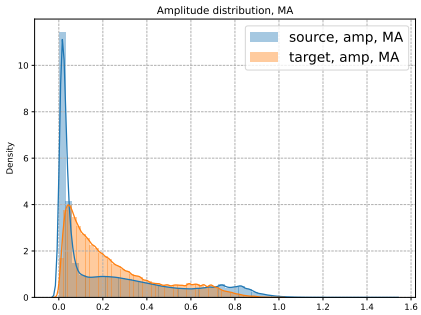
(b) Phase difference, MM



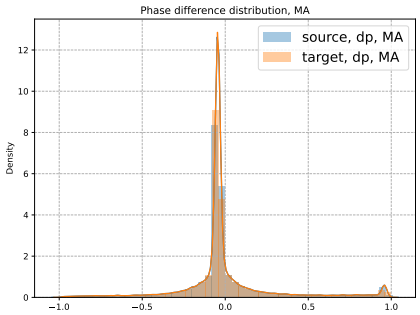
(c) Amplitude, SS



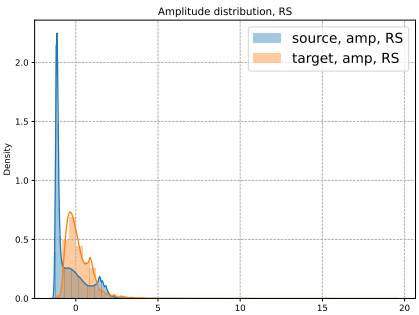
(d) Phase difference, SS



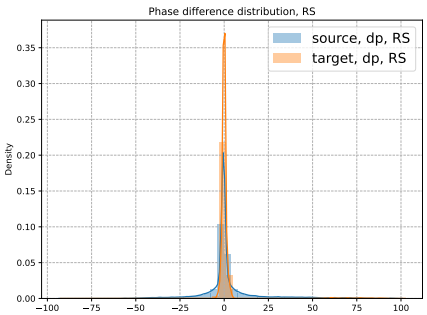
(e) Amplitude, MA



(f) Phase difference, MA



(g) Amplitude, RS

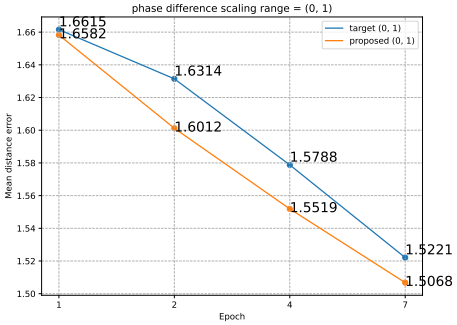


(h) Phase difference, RS

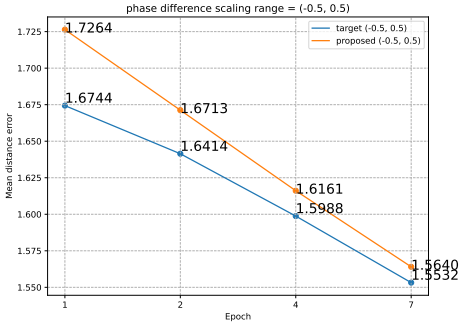
Figure 62: The distributions with different scaler

doi:10.6342/NTU202203944

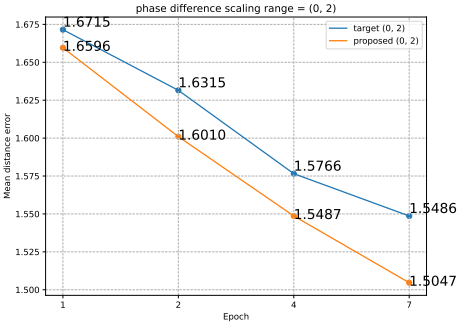
ranges are [0,1], [-0.5,0.5], [0,2], [-1,1], [0,12], and [-6,6] respectively. We find that the fine-tuning model leads to a negative transfer effect when the scaling ranges of the phase difference scaler are [-0.5,0.5],[-1,1], and [-6,6]. In conclusion, when the elements in the scaled feature data are all greater than 0, the fine-tuned model can obtain a good classification again. However, when some elements of the scaled feature data are smaller than 0, some of the model weights must be re-optimized to obtain a good classification due to the effect of ReLU.



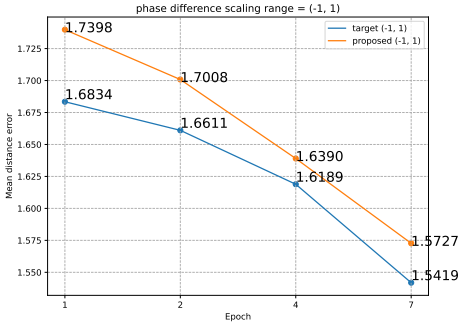
(a) MinMax, $(0, 1)$



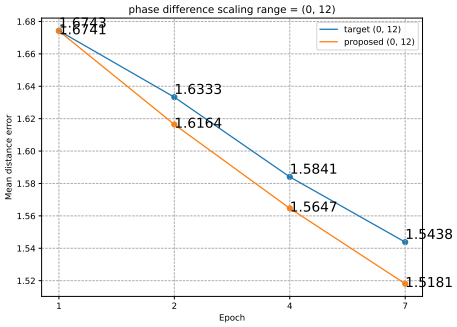
(b) MinMax, $(-0.5, 0.5)$



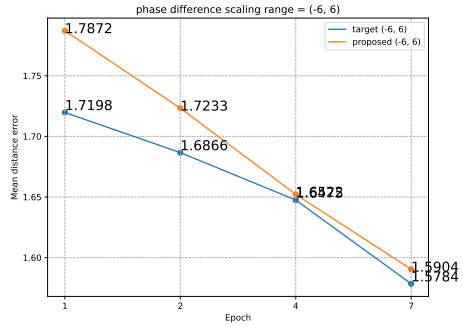
(c) MinMax, $(0, 2)$



(d) MinMax, $(-1, 1)$



(e) MinMax, $(0, 12)$



(f) MinMax, $(-6, 6)$

Figure 63: Comparing the MDE with different scaling range of MinMax scaler

We have compared the results of phase difference scaling ranges [0, 1], [0, 2], [0, 12], and these three results are similar. We did not discuss in detail the best

doi:10.6342/NTU202203944

conditions	Number of domain 3 trajectory dataset			
Model	MDE (1)	MDE (2)	MDE (4)	MDE (7)
target MICNN	1.6615	1.6314	1.5788	1.5221
fine-tuned MICNN	1.6570	1.6059	1.5521	1.5093
fine-tuned MICNN, Cons.	1.6582	1.6012	1.5519	1.5068
fine-tuned MICNN, Child-tuning	1.6591	1.6015	1.5526	1.5217

Table 24: Comparison the MDE of different fine-tuned model

conditions	Number of domain 3 trajectory dataset			
Model	MDE (1)	MDE (2)	MDE (4)	MDE (7)
target MICNN	2.2217	2.1895	2.1166	1.9900
fine-tuned MICNN	2.2126	2.1508	2.0706	1.9510
fine-tuned MICNN, Cons.	2.2091	2.1522	2.0711	1.9483
fine-tuned MICNN, Child-tuning	2.2202	2.1420	2.0721	1.9652

Table 25: Comparison the RMSE of different fine-tuned model

scaling range of the two features, so we still use the default setting of MinMax as our scaling range.

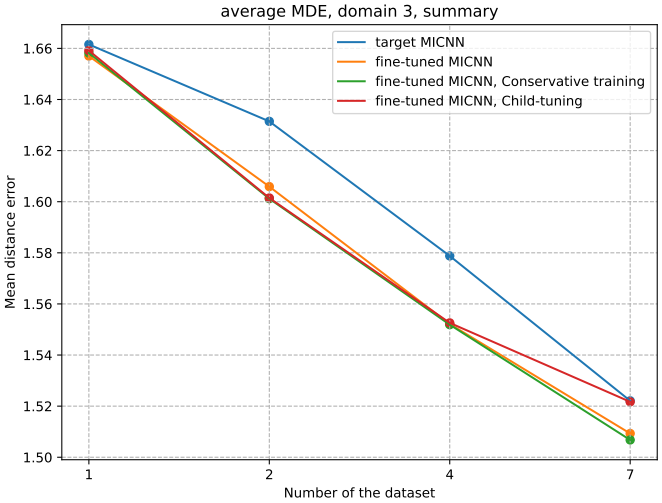
5.5 Summary and Cross Validation

Generative models do not generate simulated features that can improve the model performance at all, so we focus on summarizing the impact of the fine-tuning methods on the model performance.

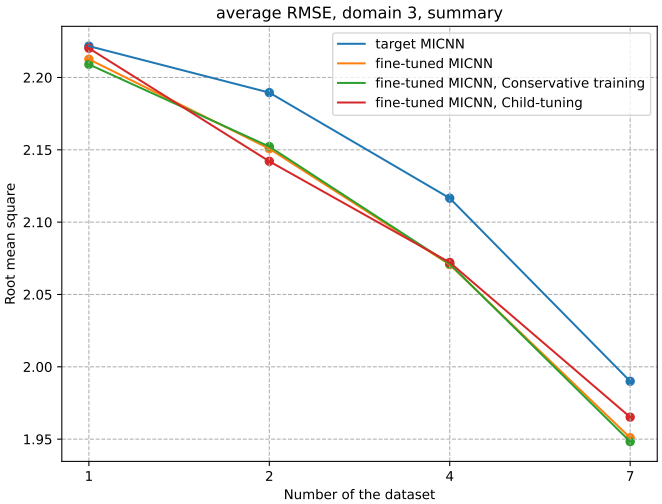
We compared three fine-tuning methods, one is to directly transfer the source model weights and fine-tune, the second is conservative training, and the third is to use Child-tuning for fine-tuning. Figure 64, 24, 25 show the comparison results of the three methods. Child-tuning is a fine-tuning method for large structures and small amounts of data, and this algorithm was proposed for the big model. Our model may not be too large in comparison, and the fine-tuned MICNN performance with the child-tuning algorithm is slightly worse than the fine-tuned model performance when the training dataset contains one and seven trajectory datasets. We use conservative training for fine-tuning, and the performance of the fine-tuned model with regularization is slightly better than the fine-tuned model performance.

In Section 4.4, We describe our detailed fine-tuned method. We used the BL521 data set to validate our proposed method and the results are shown in Figure 65. From the results of MICNN, the results of the fine-tuned model are slightly better than the performance of the target model. For MDE, comparing the target model,

doi:10.6342/NTU202203944



(a) Comparing the average MDE of three fine-tuned methods



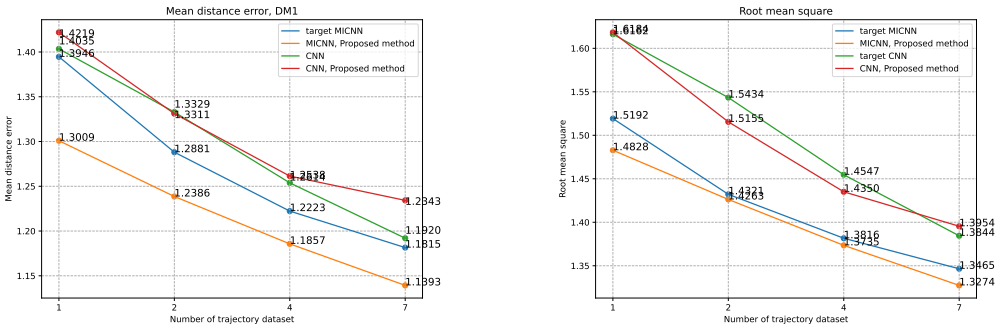
(b) Comparing the average RMSE of three fine-tuned methods

Figure 64: Comparing the average performance for fine-tuned methods

doi:10.6342/NTU202203944

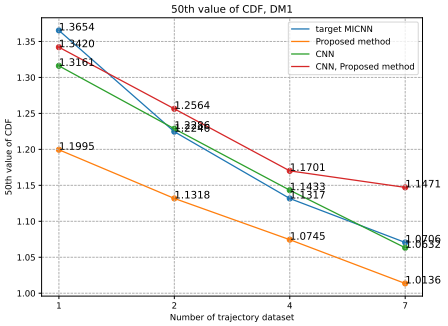
the fine-tuned model performances are improved by 6.7%, 3.8%, 3%, and 3.6%, respectively. For RMSE, the proposed model performance is slightly better than the target model performance. For MLE, the fine-tuned model performances are improved by 12.15%, 7.6%, 5.1%, and 5.3%, respectively.

From the CNN results in Figure 65, we found that our proposed method does not seem to have a significant effect. Therefore, we analyze the feature distributions of two domains in Figure 66.



(a) Comparing the MDE

(b) Comparing the RMSE



(c) 50^{th} value of CDF

Figure 65: Cross validate in domain 1 data

The amplitude distribution of domain 1 is single-peaked, and the amplitude distribution of domain 3 is bimodal in Figure 66(a).We found that when collecting BL521 data, turtlebot3 installed two antennas to receive the reference signal. When collecting data in the corridor, turtlebot3 installed one antenna, which resulted in 100 collected CSIs without antenna gain. Therefore, we designed a model using only the first hundred elements of the subcarrier amplitude as model input feature, and the result is shown in Figure 67. The CNN fine-tuned on our proposed fine-tuning algorithm is slightly better than the target model when the training data is small.

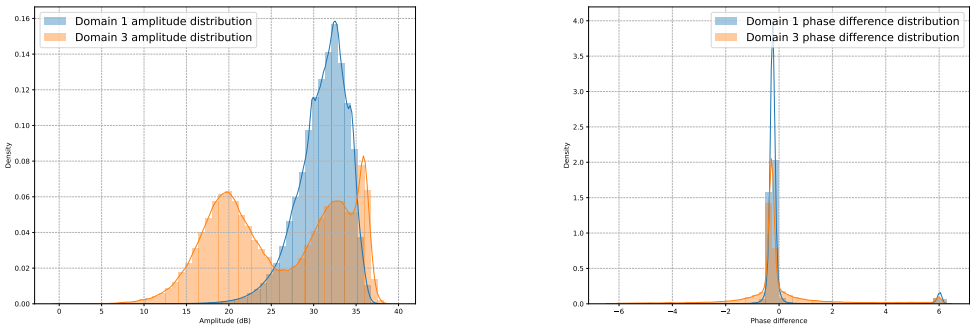
The fine-tuning method is able to improve the fine-tuned MICNN model. We believe there are several reasons why the MICNN model can be fine-tuned for

doi:10.6342/NTU202203944

improvement:

1.) Phase difference is not easily affected by the absence of antenna gain, and the knowledge of phase difference can be easily transfer to other domain model for improvement.

2.) Although the feature distribution of the two domains is somewhat different, these elements with antenna gain can still provide some useful patterns to the target domain.



(a) amplitude distributions (b) phase difference distributions

Figure 66: Comparing the distributions of domain 1 and domain 3

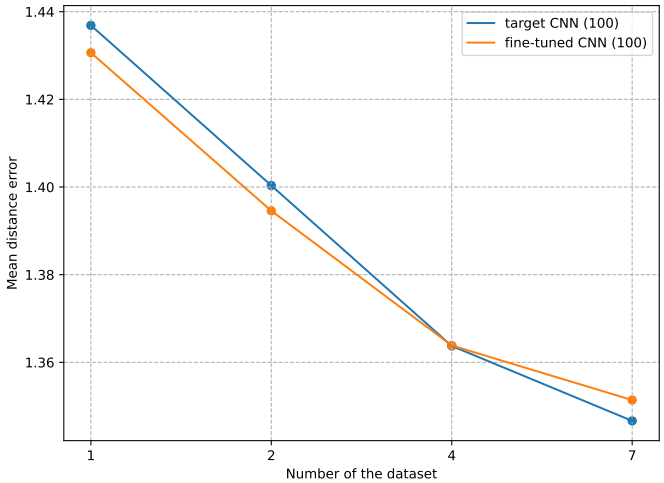


Figure 67: Comparing the MDE of the CNN train on dm1 modified feature

doi:10.6342/NTU202203944

CHAPTER 6

CONCLUSION AND FUTURE WORK

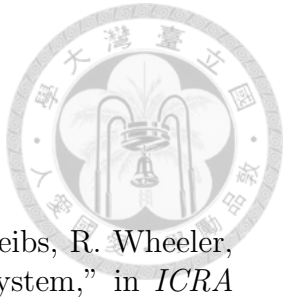
In this Section 3, We propose a CNN model using LTE CSI subcarrier amplitude and subcarrier phase difference as model input features, which outperforms the DNN model using LTE CSI subcarrier amplitude as model input. The designed MICNN-RNN model can achieve sub-meter indoor localization, and the MDE of the CNN-RNN model can reach 89 cm (In Figure 15). The performance of the CNN model we use is 12.6% better than the DNN model performance (In Figure 28) when training set is large. Comparing the CNN using subcarrier amplitude as model input feature, Our proposed MICNN model using subcarrier amplitude and subcarrier phase difference as model input features is improved by 3.3%.

In Section 4, we discuss how to use less target domain data to achieve better model performance. We applied the data augmentation approach and transfer learning approach. The data augmentation methods do not lead to better model performance compared to the model performance trained on only target domain data. Model fine-tuning is helpful to improve model performance. Compared with the model performance of the target MICNN in domain 1, the MLE and MDE of fine-tuned MICNN can improve by 12.2% and 6.7% respectively when the training set contains one trajectory dataset.

Our system is mainly based on turtlebot3 as the user equipment. The human trajectory is more complex than the turtlebot3 trajectory, and there are more changes when people move. How to use the CNN+RNN model to get better localization performance is a new challenge.

doi:10.6342/NTU202203944

REFERENCES



[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[2] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[3] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[4] R. Amsters and P. Slaets, "Turtlebot 3 as a robotics education platform," in *International Conference on Robotics in Education (RiE)*. Springer, 2019, pp. 170–181.

[5] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[7] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.

[8] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[9] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07, 2007, p. 193–200.

[10] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[11] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[12] L. Duong, T. Cohn, S. Bird, and P. Cook, "Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser," in *Proceedings of the 53rd annual meeting of the Association for Computational Linguistics and*

doi:10.6342/NTU202203944

the 7th international joint conference on natural language processing (volume 2: short papers), 2015, pp. 845–850.

[13] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

[14] M. T. Hoang, B. Yuen, X. Dong, T. Lu, R. Westendorp, and K. Reddy, "Recurrent neural networks for accurate rssi indoor localization," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 639–10 651, 2019.

[15] S. Bai, M. Yan, Q. Wan, L. He, X. Wang, and J. Li, "Dl-rnn: An accurate indoor localization method via double rnns," *IEEE Sensors Journal*, vol. 20, no. 1, pp. 286–295, 2019.

[16] H. Zhang, Z. Zhang, S. Zhang, S. Xu, and S. Cao, "Fingerprint-based localization using commercial lte signals: A field-trial study," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. IEEE, 2019, pp. 1–5.

[17] E. Schmidt, D. Inupakutika, R. Mundlamuri, and D. Akopian, "Sdr-fi: Deep-learning-based indoor positioning via software-defined radio," *IEEE Access*, vol. 7, pp. 145 784–145 797, 2019.

[18] X. Peng, R. Chen, K. Yu, F. Ye, and W. Xue, "An improved weighted k-nearest neighbor algorithm for indoor localization," *Electronics*, vol. 9, no. 12, p. 2117, 2020.

[19] Y. Zhang, C. Qu, and Y. Wang, "An indoor positioning method based on csi by using features optimization mechanism with lstm," *IEEE Sensors Journal*, vol. 20, no. 9, pp. 4868–4878, 2020.

[20] X. Wang, L. Gao, and S. Mao, "Csi phase fingerprinting for indoor localization with a deep learning approach," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1113–1123, 2016.

[21] G. Pecoraro, S. Di Domenico, E. Cianca, and M. De Sanctis, "CSI-based fingerprinting for indoor localization using LTE signals," *EURASIP Journal on Advances in Signal Processing*, vol. 2018, no. 1, p. 49, 2018.

[22] Y. Yin, X. Yang, P. Li, K. Zhang, P. Chen, and Q. Niu, "Localization with transfer learning based on fine-grained subcarrier information for dynamic indoor environments," *Sensors*, vol. 21, no. 3, p. 1015, 2021.

[23] K. Liu, H. Zhang, J. K.-Y. Ng, Y. Xia, L. Feng, V. C. Lee, and S. H. Son, "Toward low-overhead fingerprint-based indoor localization via transfer learning: Design, implementation, and evaluation," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 898–908, 2017.

[24] C. Wu, Z. Yang, Y. Liu, and W. Xi, "Will: Wireless indoor localization without site survey," *IEEE Transactions on Parallel and Distributed systems*, vol. 24, no. 4, pp. 839–848, 2012.

doi:10.6342/NTU202203944

[25] Q. Liang and M. Liu, "An automatic site survey approach for indoor localization using a smartphone," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 1, pp. 191–206, 2019.

[26] A. Belmonte-Hernández, G. Hernández-Peñaloza, D. M. Gutiérrez, and F. Álvarez, "SWiBluX: Multi-Sensor Deep Learning Fingerprint for precise real-time indoor tracking," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3473–3486, 2019.

[27] X. Wang, C. Yang, and S. Mao, "Phasebeat: Exploiting csi phase data for vital sign monitoring with commodity wifi devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1230–1239.

[28] Y. Xie, Z. Li, and M. Li, "Precise power delay profiling with commodity wifi," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1342–1355, 2018.

[29] M. Speth, S. A. Fechtel, G. Fock, and H. Meyr, "Optimum receiver design for wireless broad-band systems using ofdm. i," *IEEE Transactions on communications*, vol. 47, no. 11, pp. 1668–1677, 1999.

[30] "feature selection algorithm:mutual information classify," https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html#sklearn.feature_selection.mutual_info_classif.

[31] "feature selection algorithm:mutual information regression," https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_regression.html#sklearn.feature_selection.mutual_info_regression.

[32] , "An indoor localization system for cellular networks based on transfer learning with reduced cost on site survey / jun-xiang liao," 2021.

[33] W. Njima, M. Chafii, and R. M. Shubair, "Gan based data augmentation for indoor localization using labeled and unlabeled data," in *2021 International Balkan Conference on Communications and Networking (BalkanCom)*. IEEE, 2021, pp. 36–39.

[34] W. Njima, A. Bazzi, and M. Chafii, "Dnn-based indoor localization under limited dataset using gans and semi-supervised learning," *IEEE Access*, vol. 10, pp. 69 896–69 909, 2022.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[37] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, "Deep generative stochastic networks trainable by backprop," in *International Conference on Machine Learning*. PMLR, 2014, pp. 226–234.

doi:10.6342/NTU202203944

[38] G. E. Hinton, "Boltzmann machine," *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007.

[39] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[40] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *Advances in neural information processing systems*, vol. 29, 2016.

[41] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*. PMLR, 2017, pp. 2642–2651.

[42] M. Nabati, H. Navidan, R. Shahbazian, S. A. Ghorashi, and D. Windridge, "Using synthetic data to enhance the accuracy of fingerprint-based localization: A deep learning approach," *IEEE Sensors Letters*, vol. 4, no. 4, pp. 1–4, 2020.

[43] H. yi Lee. Ml lecture 19: Transfer learning. Online Available at: https://youtu.be/qD6iD4TFsdQ

[44] R. Aljundi, "Continual learning in neural networks," *arXiv preprint arXiv:1910.02718*, 2019.

[45] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[46] D. J. MacKay, "A practical bayesian framework for backpropagation networks," *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[47] R. Xu, F. Luo, Z. Zhang, C. Tan, B. Chang, S. Huang, and F. Huang, "Raise a child in large language model: Towards effective and generalizable fine-tuning," *arXiv preprint arXiv:2109.05687*, 2021.

[48] W. Zhang, L. Deng, L. Zhang, and D. Wu, "A survey on negative transfer," *arXiv preprint arXiv:2009.00909*, 2020.

[49] M. M. Ahsan, M. P. Mahmud, P. K. Saha, K. D. Gupta, and Z. Siddique, "Effect of data scaling methods on machine learning algorithms and model performance," *Technologies*, vol. 9, no. 3, p. 52, 2021.

[50] Q. Li, H. Qu, Z. Liu, N. Zhou, W. Sun, S. Sigg, and J. Li, "Af-dcgan: Amplitude feature deep convolutional gan for fingerprint construction in indoor localization systems," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 3, pp. 468–480, 2019.

[51] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

doi:10.6342/NTU202203944

[52] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[53] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, "To transfer or not to transfer," in *In NIPS'05 Workshop, Inductive Transfer: 10 Years Later*, 2005.

doi:10.6342/NTU202203944