Project No: B3285-232

DEEP LEARNING-BASED ESTIMATION OF WALL PARAMETERS FOR THROUGH-THE-WALL IMAGING

Submitted by: Christian Joseph

Supervisor: Assistant Professor Abdulkadir C. Yucel

School of Electrical & Electronic Engineering

A final year project report presented to Nanyang Technological University
in partial fulfilment of the requirements for the
Degree of Bachelor of Engineering

2024

Table of Contents

Abstract

This project explores the use of deep learning algorithms, particularly Convolutional Neural Networks (CNNs), to estimate wall parameters in Through-the-Wall Imaging (TWI). TWI utilizes Ground Penetrating Radar (GPR) to detect objects hidden behind walls, but distinguishing between the target and clutter from the wall can be quite challenging. Traditional signal processing techniques often struggle with complex wall structures, which results to detection inaccuracies.

This project uses GPRMax software, an open-source radar simulation software, to create a dataset of synthetic B-scan images. CNN models are then developed using TensorFlow, which are trained on different datasets to learn the relationship between the B-scan data and simulation parameters. This project begins by estimating wall parameters and then expands to include the prediction of object parameters such as position and permittivity.

List of Figures

List of Tables

Chapter 1 Introduction

This chapter provides an overview of the FYP, including background of the project, the objectives and scope of the project, and the structure of the report.

1.1 Background

Through the wall imaging (TWI) is a technique to locate objects behind walls or barriers. In a TWI setup, a Ground Penetrating Radar (GPR) will emit electromagnetic waves towards a wall. These waves will pass through the wall and interact with any materials behind it. Upon contact with different materials, the waves are partially reflected, depending on the material's properties such as permittivity, thickness, and conductivity. A receiver positioned near the transmitter captures these reflected waves. By analyzing the characteristics of the reflected waveforms, the object behind the wall can be identified.

However, one of the major challenges in TWI is distinguishing between objects from clutter caused by the wall. Walls often create significant interference, as electromagnetic waves reflect off them before reaching the object. This clutter can distort the object's signal and reduce the detection accuracy. However, the clutter can be minimized if the wall properties, such permittivity, and thickness are known. This will lead to a more accurate object detection.

Traditional signal processing methods usually struggle when having to differentiate between reflected waveforms from the target object and those reflected by the wall, especially in cases with complex wall structures or varying material properties. These challenges can significantly reduce the accuracy of object detection Through-the-Wall Imaging (TWI) cases.

On the other hand, deep learning offers a more effective solution. Deep neural networks can be trained on large datasets of radar images paired with known wall parameters. Once

trained, these models can efficiently estimate the wall characteristics such as thickness and permittivity. Once integrated into TWI systems, these models can significantly improve the object detection and minimizing the impact of clutter from the wall.

1.2 Objectives and Scope

This project aims to improve TWI capabilities by applying deep learning algorithms to estimate wall parameters. The primary goal is to develop a Convolutional Neural Network (CNN) model that can accurately predict key simulation parameters.

The model development process will be conducted in three stages. The first stage will focus on estimating the wall parameters alone, such as thickness and permittivity. In the second stage, the model will be extended to estimate both wall parameters and object position. Finally, in the third stage, the model will be trained on datasets with randomly positioned objects behind the wall and varied object properties. This approach will enable the model to predict both the wall and object properties. By including different features, these models can distinguish the object from the clutter more effectively.

The scope of this project includes these following key areas:

1. **Data Generation and Preparation:**

   The initial phase involves generating a comprehensive dataset of radar images and their corresponding wall and object parameters. Real-world data is impractical for training deep learning models at scale, so synthetic B-scan radar images will be generated in this project. This project leverages GPRMax, an open-source radar simulation software. GPRMax is a powerful tool that simulates the propagation of electromagnetic waves through various materials, allowing for the modelling of different wall configurations [1].

After generation of the synthetic radar data, relevant features will be extracted, and each B-scan image will be labelled with corresponding parameters. The processed and consolidated dataset will then be used to train the deep learning model.

2. Deep Learning Model (CNN) Design and Validation:

After preparation of the dataset, the next step is to design the Convolutional Neural Network (CNN) to predict the wall parameters based on the B-scan data. CNNs are well-suited for this task due to their ability to learn meaningful patterns from image data. To build and train the CNN, this project will use TensorFlow, an open-source machine learning framework from Google [2]. TensorFlow is widely known for its efficiency, which makes it a popular choice for machine learning projects.

The CNN model will analyse synthetic B-scan images generated by GPRMax to estimate simulation parameters, such as wall thickness and permittivity. In this context, regression techniques will be used to predict continuous output values. By training the model on the labelled data, the model can learn meaningful patterns between B-scan images and their corresponding parameters, which allows it to make accurate predictions on the test data. To evaluate the model's performance, the error between the predicted and actual values will be assessed.

1.3 Organisations

The content of this report is structured into the following chapters:

Chapter 2: Literature Review

This chapter will provide an overview of existing research and methods to estimate wall parameters in TWI. It will summarize key findings in wall parameters estimation and highlight the importance of deep learning techniques.

Chapter 3: Generation of a B-scan Data

This chapter will explore the process of generating B-scan images for Through-the-Wall Imaging (TWI) configuration and include an analysis of the B-scan data alongside theoretical calculations.

Chapter 4: B-Scan Dataset Generation and Preparation with Python

This chapter will explain how to generate three different B-scan datasets using Python in Jupyter Notebook. It will cover the automation of setup creation, running GPRMax simulations, extracting results, compiling output files, and organizing dataset labels to ensure each data point is associated with its corresponding wall parameters in an organized file. This chapter will be divided into three sections, each to create datasets with different varying parameters: (1) wall parameters only, (2) wall parameters and object position, and (3) wall parameters, object position, and permittivity.

Chapter 5: Convolutional Neural Network (CNN) Model Development and Evaluation

This chapter will detail the implementation of CNN for estimating wall and object parameters using three different datasets. It will cover the CNN architecture, describe the training parameters and processes, evaluate each model's prediction accuracy, and analyze which datasets or parameters can be predicted accurately by the model.

Chapter 6: Conclusion and Future Work

This chapter will provide a summary of the findings from the project and evaluate the CNN model for wall parameters estimation. This section will also address the challenges and suggest improvements for future research.

Chapter 2 : Literature Review

Over the years, numerous methods have been developed for estimating the wall parameters in the context of Through-the-Wall Imaging (TWI) using Ground Penetrating Radar (GPR). This chapter reviews approaches for wall parameter extraction in TWI scenarios.

Traditional methods for estimating wall parameters primarily focus on time-delay measurements of reflected electromagnetic waves. By analyzing these time delays, researchers can infer important characteristics of the wall, such as thickness and permittivity. However, the effectiveness of these methods can be significantly impacted by various factors, including the complexity of the wall structure, the distance between the transmitter and the receiver, and the presence of overlapping reflections.

Time-Delay Estimation (TDE) methods analyze reflected waveforms based on time delays, which enables the estimation of wall thickness and permittivity from measurements taken at different transmitter and receiver distances. Employing TDE methods in the frequency domain, particularly through subspace super-resolution methods, can help to resolve overlapping reflections even in complex conditions, such as those encountered with limited radar bandwidth [3]. Focusing on TDE methods can enhance wall parameter extraction accuracy and improve object detection capabilities, which makes it suitable for TWI applications.

While traditional methods have demonstrated effectiveness, real-world scenarios often present greater complexity. Deep learning approaches offer a solution to these challenges. By training deep learning algorithms on large datasets and using advanced neural network architectures, these models can identify complex patterns in the data and handle the variability in wall structures more effectively, which could lead to improved accuracy in estimating the wall parameters. In this project, deep learning methods will be explored in simple TWI scenarios to evaluate their effectiveness for simulation parameters estimation.

Chapter 3 : Generation of a B-Scan Data

This section outlines the process of generating a B-scan data using GPRMax software. It covers the steps involved in modelling the TWI setup, creating the GPRMax script, running the simulation, generating the B-scan data, and analyzing the resulting B-scan data.

3.1 Modeling Setup for TWI Simulation

The setup for TWI simulation consists of a homogeneous wall with specified parameters such as thickness and permittivity, and a reference object positioned behind the wall. The wall acts as a barrier, which reflects and scatters the wave.

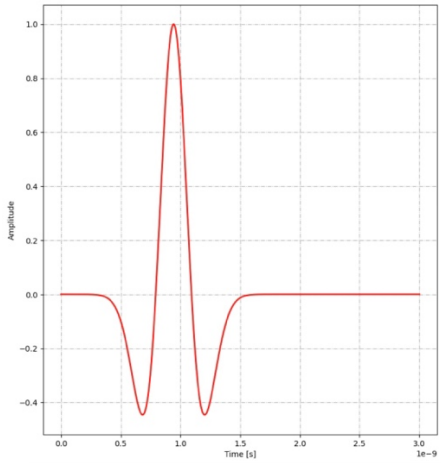


Figure 1

A GPR system, which consists of a transmitter and a receiver, is positioned at a certain distance in front of the wall. The transmitter emits electromagnetic waves that penetrate through the wall, while the receiver captures the reflected signals. In this project, a Hertzian dipole is used to represent the transmitter and receiver due to its simplicity in modelling the radiation pattern of a small point source. The Ricker waveform is employed as the transmitted signal, which is commonly used in GPR applications. This waveform is ideal for modelling the propagation of electromagnetic waves through different materials and effective in detecting changes in subsurface features like walls and hidden objects.

In this project, hertzian dipole and ricker waveform is used instead of antenna models available in GPRMax due to their greater computational efficiency. While GPRMax's antenna models provide detailed simulations, they are resource-intensive, requiring significant memory and processing power. Generating a B-scan takes significantly more time, which makes them impractical for this project's requirement to produce large datasets efficiently.

To simulate an open space, perfectly matched layers (PML) are applied at the boundaries of the simulation domain. PML effectively absorbs electromagnetic waves and prevents reflections from the edges. It ensures that the waveform from the transmitter do not reflect from the edges of the domain, which could impact the accuracy of the B-scan.

The reflected waveform captured by the receiver forms an A-scan, which is a one-dimensional trace of signal received by the GPR from a specific position. In contrast, a B-scan is a two-dimensional signal data formed by multiple A-scans collected at different positions. To generate the B-scan, the transmitter and receiver are moved horizontally along the wall at a fixed interval. It will capture a series of A-scans at different locations along the horizontal line.

To accurately capture the reflected waves in the B-scan, an appropriate time window must be set in the simulation. The time window should be sufficiently long to capture all the reflected waves in the B-scan. The minimum travel time can be calculated based on the distance the electromagnetic wave travels, which is the time taken for the wave to transmit from the transmitter and reflected to the receiver. This ensures all relevant reflections are recorded within the time window.

Another important parameter in the simulation is the spatial resolution. Spatial resolution refers to the smallest feature in the simulation, like a pixel in an image. A higher spatial resolution provides more detailed results, but it also increases the computational load and processing time.

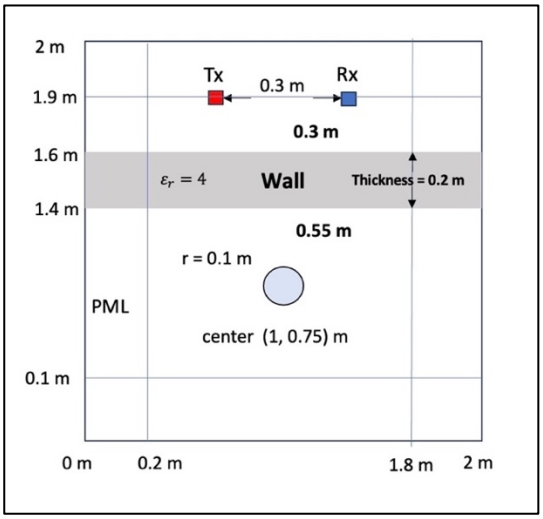


Figure 2

The setup illustrated in figure 2 will be used to demonstrate the sample Through-the-Wall Imaging (TWI) simulation in GPRMax. The simulation domain size is 2 m x 2 m, and the radar is placed 0.3 m in front of the wall. The transmitter emits ricker waveform with central frequency of 1 GHz. The distance between the transmitter and the receiver is 0.2 m. The wall thickness is 0.2 m, and the relative permittivity of the wall is 4. The object used in this simulation is a perfect electrical conductor (PEC), which allows the waveform to be completely reflected because no wave can penetrate a conductor. The object is cylindrical, with a radius of 0.1 m, and is located at the center of the room, 0.55 m behind the wall.

According to the GPRMax documentation, the spatial resolution should be at least ten times smaller than the smallest wavelength in the simulation. The smallest wavelength is determined by the highest frequency and the slowest speed of the wave, which occurs when the wave is travelling in the wall. The minimum wavelength can be calculated using the equation:

$$\lambda_{min} = \frac{v_{min}}{f_{max}}$$

The speed of the electromagnetic wave in a medium is determined by its relative permittivity. A higher permittivity means that the wave will travel slower inside the

medium. The speed of a waveform in a medium can be calculated by:

$$v = \frac{c}{\sqrt{\varepsilon_r}}$$

The highest frequency in the simulation is obtained by analyzing the spectrum of the waveform. In this case, by utilizing GPRMax tools, the highest estimated significant frequency is 2.797 GHz. The minimum wavelength can be calculated as:

$$\lambda_{min} = \frac{c}{f_{max}\sqrt{\varepsilon_r}} = \frac{3 \times 10^8}{2.79 \times 10^9 \sqrt{4}} = 0.05\ m$$

Since the λ_{min} is 0.05 m, the spatial resolution will be set to 0.005 m. Next, the Perfectly Matched Layer (PML) for this simulation is set to be 20 cells or 0.1 m for all sides.

The minimum travel time should be calculated to determine the time window of the simulation. It can be calculated by considering d, the distance between the transmitter and the object and v_{min}, slowest wave speed in the simulation.

$$t_{min} = \frac{2d}{v_{min}} = \frac{2 \times 1.05}{\frac{3 \times 10^8}{\sqrt{4}}} = 1.4 \times 10^{-8}\ s$$

Based on calculation, the time window of the simulation must be greater than the $1.4 \times 10^{-8}\ s$ to ensure that the reflected waves are captured. Once all the parameters are determined, the simulation can proceed.

3.2 Running TWI Simulation in GPRMax

An input script is required to set up the TWI simulation in GPRMax. This file consists of all the necessary parameters, formatted according to GPRMax template, which defines the materials and configuration of the simulation. Figure 3 shows the input script for the setup in figure 2.

```
#GPRMax Input Script
template = """
#title: TWI Simulation
#domain: 2 2 0.005
#dx_dy_dz: 0.005 0.005 0.005
#time_window: 1.75e-8

#material: {permittivity} 0 1 0 wall

#waveform: ricker 1 1e9 rickerwave

#hertzian_dipole: z 0.1 1.9 0 rickerwave
#rx: 0.4 1.9 0

#src_steps: 0.1 0 0
#rx_steps: 0.1 0 0

#box: 0 1.4 0 2 1.60 0.005 wall
#cylinder: 1 0.75 0 1 0.75 0.005 0.1 pec
#pml_cells: 20 20 0 20 20 0
"""
```

Figure 3

GPRMax provides tools to generate and plot B-scan images. For B-scan generation, the input script must specify the step size of both the transmitter (*src_steps*) and receiver (*rx_steps*). The number of steps (*n*) is specified when running the simulation. In this example, the *n* is set to 15.

To create the B-scan image for this simulation, the GPR will be moved horizontally in 15 steps with 0.1 m each step. Usually, a more detailed B-scan image would require higher number of A-scans with smaller step sizes. However, due to time constraints, the simulation is simplified to speed up the process. Once the simulation is processed, the resulting numerical data can be plotted using GPRMax's tool. Figure 4 displays the B-scan data generated from the simulation.

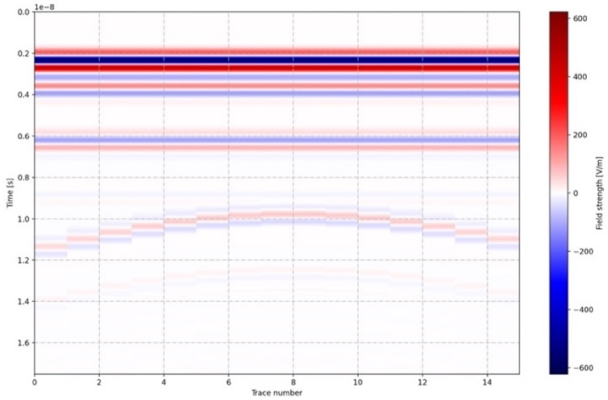


Figure 4

3.3 Analysis of B-scan Data

In this section, theoretical calculation will be performed and compared with the components in the sample B-scan result (from figure 4). The B-scan image includes two main components: the wall and the object. To identify the reflections from the wall, reflection from both front and back sides must be considered.

Figure 5 illustrate the wave reflection when the GPR is positioned at the center. Upon hitting the wall surface, a portion of the electromagnetic waves is reflected, while the rest penetrate through the wall. At the back of the wall, some waves will be reflected, while others continue through to the other side.

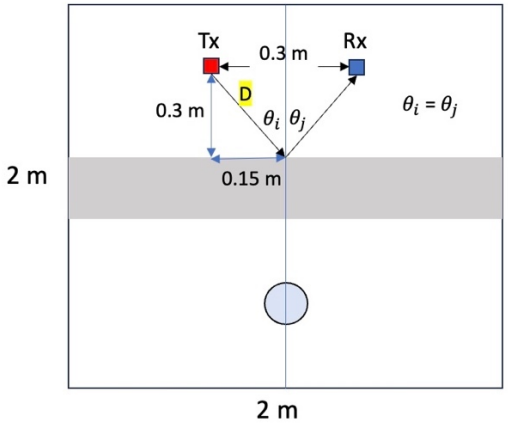


Figure 5

The time of reflection from the front side of the wall can be calculated using the law of reflection. According to this principle, the angle of incidence is equal to the angle of reflection. The time of reflection from the front side can be calculated:

$$D = \sqrt{0.3^2 + 0.15^2} = 0.335\ m$$

$$T_{front} = \frac{2D}{c} = \frac{0.670}{3 \times 10^8} = 2.233\ ns$$

The waves will travel from air to wall before reflecting off the back side. According to

Snell's law, when a wave crosses different medium, the angle of transmission will be influenced by the permittivity of the materials. As the waves enter the wall, they will slow down due to higher permittivity, which alters their angle of propagation in the medium. The illustration of the wave path is shown in figure 6.

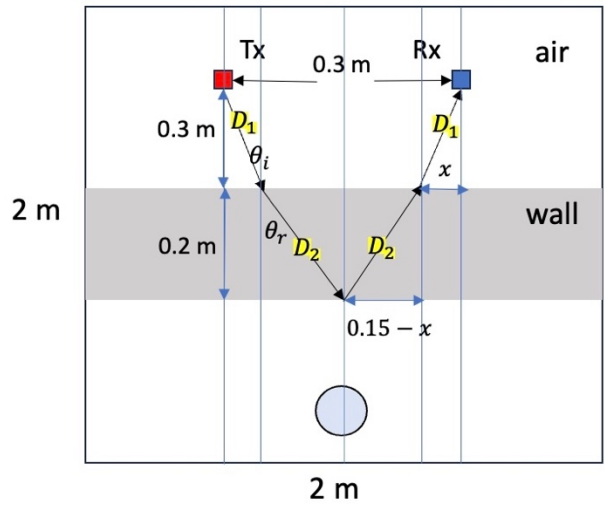


Figure 6

The Snell's law is given by:

$$\frac{\sin(\theta_i)}{\sin(\theta_r)} = \frac{\sqrt{\varepsilon_2}}{\sqrt{\varepsilon_1}}$$

By using Snell's law, the distances D_1 and D_2 can be calculated:

$$\sin(\theta_i) = 2\sin(\theta_r)$$

$$\frac{x}{\sqrt{0.3^2 + x^2}} = 2\frac{(0.15 - x)}{\sqrt{(0.15 - x)^2 + 0.2^2}}$$

$$D_1 = \sqrt{x^2 + 0.3^2} = 0.321\ m$$

$$D_2 = \sqrt{(0.15 - x)^2 + 0.2^2} = 0.203\ m$$

Once the distances are determined, the total time to observe the reflection from the back side can be calculated. The speed of wave inside the wall will be slower due to higher

permittivity.

$$v_2 = \frac{v_1\sqrt{\varepsilon_1}}{\sqrt{\varepsilon_2}} = 1.5 \times 10^8\ m/s$$

$$T_{back} = \frac{2D_1}{v_1} + \frac{2D_2}{v_2} = \frac{2 \times\ 0.321}{3 \times 10^8} + \frac{2 \times\ 0.203}{1.5 \times 10^8} = \ 4.846\ ns$$

The other component of the B-scan corresponds to the wave reflection from the object behind the wall. Since the object in the simulation is a PEC, the wave is fully reflected without penetrating it.

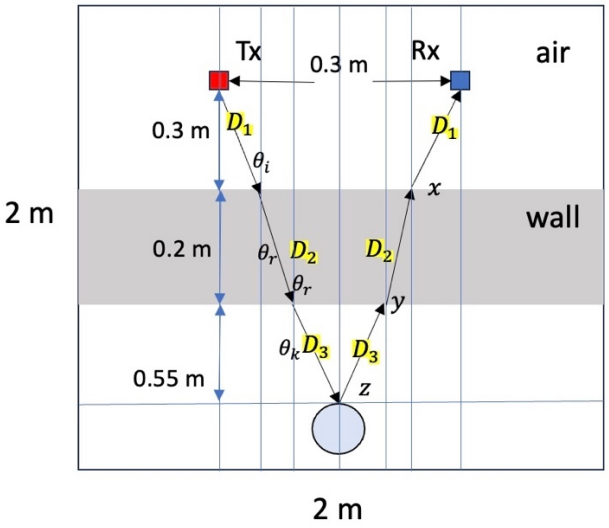


Figure 7

Similarly, using Snell's law, the geometric relationship of the wave path can be inferred:

$$\frac{\sin\theta_i}{\sin\theta_r} = \frac{\sqrt{\varepsilon_2}}{\sqrt{\varepsilon_1}}\ ;\ \frac{\sin\theta_r}{\sin\theta_k} = \frac{\sqrt{\varepsilon_1}}{\sqrt{\varepsilon_2}}$$

$$\sin\theta_i = 2\sin\theta_r\ ;\ \ \sin\theta_k = 2\sin\theta_r\ \ ;\ \sin\theta_i = \ \sin\theta_k$$

$$\frac{x}{\sqrt{x^2+0.3^2}} = \ 2\ \frac{y}{\sqrt{y^2+0.2^2}} = \frac{z}{\sqrt{z^2+0.55^2}}\ \ ;\ \ x + y + z = 0.15$$

Using this relationship, the values of x, y, z can be obtained. Subsequently, the distances traveled by the wave ($D_1,\ D_2,\ D_3$) can be calculated as well.

$$x = 0.0474\,m\,;\; y = 0.0156\,m\,;\; z = 0.0869\,m$$
$$D_1 = 0.3037\,m\,,\; D_2 = 0.2006\,m\,,\; D_3 = 0.5568\,m$$

Once the distances are known, the time traveled by the wave can be estimated:

$$T_{object} = \frac{2D_1}{v_1} + \frac{2D_2}{v_2} + \frac{2D_3}{v_3} = \frac{2(0.3037)}{3 \times 10^8} + \frac{2(0.2006)}{1.5 \times 10^8} + \frac{2(0.5568)}{3 \times 10^8} = 8.412\,ns$$

To determine the total time delay for reflections in the B-scan image, both the travel time and duration of the wave must be considered. In this simulation, a ricker waveform with a frequency of 1 GHz is utilized. By plotting this waveform in GPRMax, the estimated duration is 1.75 ns as shown in figure below.

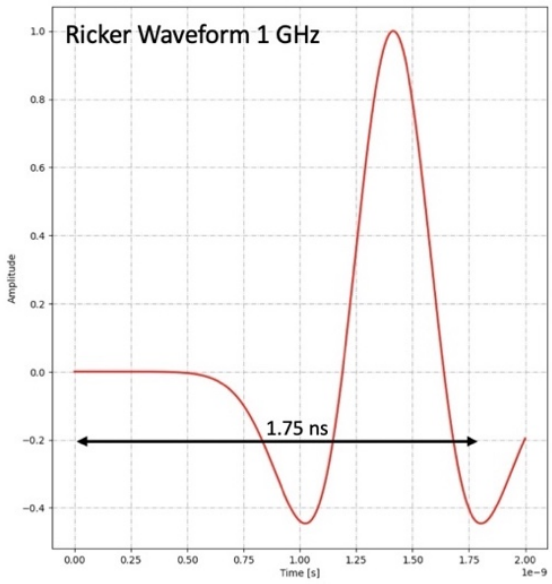


Figure 8

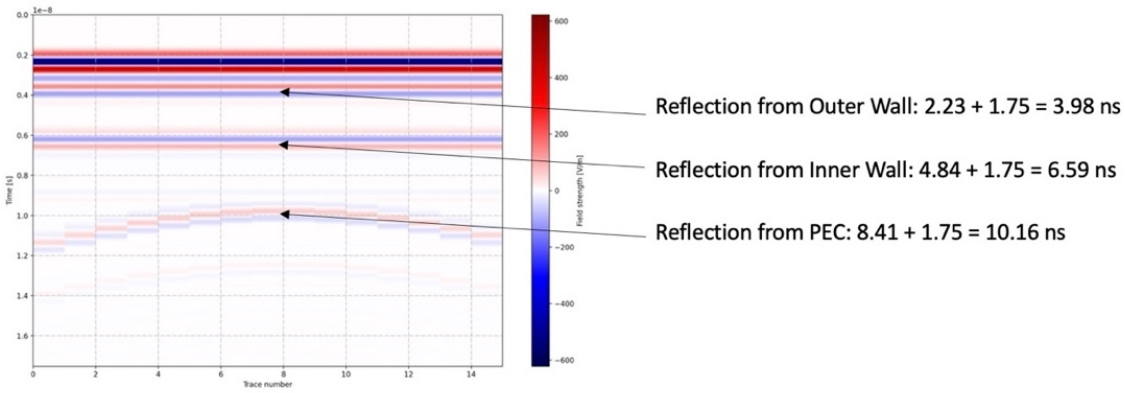


Figure 9

By comparing the calculated time delays with the B-scan data, it can be confirmed that each component in the B-scan plot corresponds accurately with the calculated time, as shown in figure 9. The reflection from the wall appears as a straight line due to the flat surface of the wall. In contrast, the reflection from the object (PEC) is hyperbolic because of the object's cylindrical shape.

Chapter 4 :B-Scan Dataset Generation and Preparation with Python

This section focuses on generating TWI simulations for three different scenarios. The objective is to create a comprehensive dataset for training machine learning models by automating the simulation processes using Python. This approach allows for efficient handling of multiple parameters to ensure the dataset is suitable for training.

4.1. Generating B-Scan Dataset for Wall Parameters Estimation (First Model)

To generate the B-scan dataset, the simulation must cover various parameters. In this first simulation, multiple TWI scenarios will be created, focusing on two key wall parameters: thickness and relative permittivity. The simulation generation process is automated using Python scripts in Jupyter Notebook.

The initial step involves setting up a model template for the GPRMax simulations, which includes the corresponding script setup. The figure below illustrates the base template for the simulations related to the first dataset, highlighting the variables that will be adjusted for different scenarios.

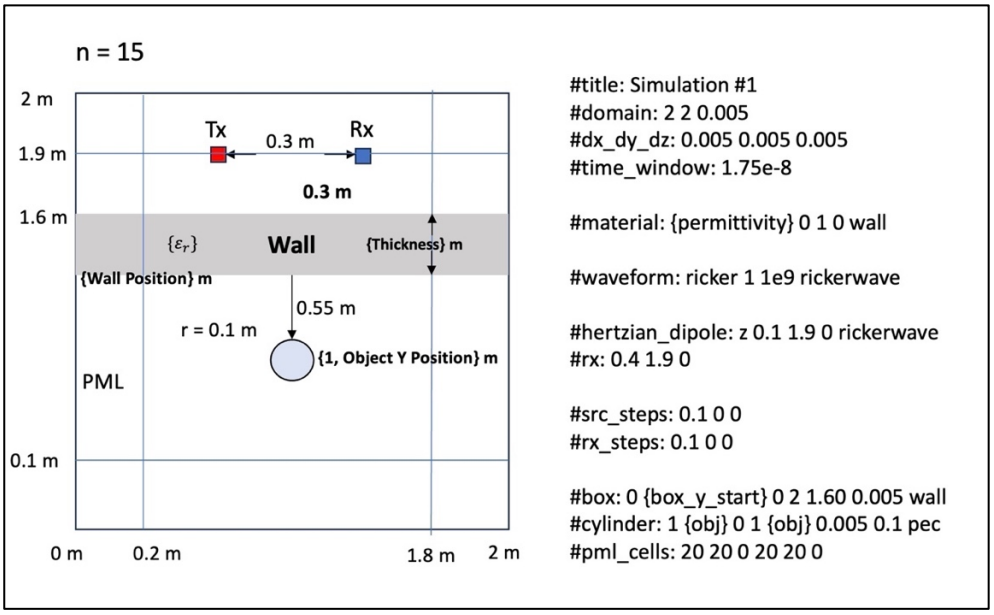


Figure 10

For the first dataset, the wall thickness ranges from 0.15 m to 0.5 m, with increments of 0.025 m. The relative permittivity ranges from 2 to 6, with increments of 0.25. A list is then created to store the combinations of wall thickness and permittivity values. This list includes all possible combinations, resulting in 255 unique data, with 15 thickness and 17 permittivity values.

Wall_parameters = [(thickness1, permittivity1), (thickness2, permittivity2), ... (thickness255, permittivity255)]

To streamline the simulation process, several functions are defined to facilitate automated processing. The first function is called *create_input_file*. This function takes the wall thickness and permittivity as parameters and calculates the position of the wall and object based on its thickness. In this simulation, the distance from the wall to the object is fixed to 0.55 m.

Each input file is named according to its index to ensure each simulation is uniquely identified. The content of the input file is then generated using a predefined template, which incorporates the permittivity, wall position, and object position. This approach allows for the efficient creation of all the input files for the simulations.

```
def create_input_file(thickness, permittivity, index):
    wall_position = 1.6 - thickness
    obj_position = wall_position - 0.55 - 0.10

    filename = os.path.join(input_dir, f'simulation_{index}bscan.in')
    output_file = os.path.join(input_dir, f'simulation_{index}bscan')

    content = template.format(
        permittivity=permittivity,
        wall_position=wall_position,
        obj_position = obj_position

    with open(filename, 'w') as f:
        f.write(content)
    return filename, output_file, f'simulation_{index}bscan_merged.out'
```

Figure 11

After creating the input files, the next step will involve running each simulation and generate the B-scans. The *run_simulation* function will be utilized to generate *n* steps A-scans for each input file, while the *merge_output_files* function is used to combine these A-scans into B-scans. In this simulation, the value of *n* is 15.

```
def run_simulation(input_file, n_iterations):
    cmd = f'python -m gprMax {input_file} -n {n_iterations}'
    subprocess.run(cmd, shell=True)

def merge_output_files(output_file):
    cmd = f'python -m tools.outputfiles_merge {output_file}'
    subprocess.run(cmd, shell=True)
```

Figure 12

The output of the B-scan files from GPRMax is stored in HDF5 file format. These files contain arrays that represent the time history of electric field and magnetic field components at the receiver position. In this simulation, the E_Z component is crucial because it represents the electric field's vertical component, which is essential for analyzing the waves interaction with the wall and object. To extract the E_Z component from the output file, the *extract_ez* function is created. This function opens the HDF5 file and retrieves E_Z field data from the specified path *rxs/rx1/Ez* from the *.out* file.

```
def extract_ez(file_path):
    try:
        with h5py.File(file_path, 'r') as file:
            Ez = file['rxs/rx1/Ez'][:]
            return Ez

def move_merged_file(output_file, index):
    merged_file = f'{output_file}_merged.out'
    Ez = extract_ez(merged_file)
    if Ez is not None:
        ez_filename = f'simulation_{index}bscan_Ez.npy'
        ez_output_path = os.path.join(ez_output_dir, ez_filename)
        np.save(ez_output_path, Ez)
    return ez_filename
```

Figure 13

The *move_merged_file* function is employed to manage the merged output file generated from the B-scan simulations with GPRMax. This function constructs the filename for the merged output file and calls the *extract_ez* function to retrieve the E_Z data, which is then saved as a NumPy array in a designated folder. This folder will contain all the NumPy files (.npy) with the E_Z data from each B-scan result and will be used for training. NumPy arrays are used as the data format because they are optimized for numerical computation and provide efficient storage as compared to standard Python list. Additionally, NumPy has also seamless integration with machine learning libraries like TensorFlow and PyTorch, which allows effective training with the machine learning algorithms.

To estimate the wall parameters, accurate labels are essential during the training process. These labels consist of the simulation parameters, specifically the wall thickness and permittivity values associated with each E_Z data file. By accurately labelling the data, the machine learning algorithms can learn the relationship between the input features (the E_Z data) and the output parameters (the wall thickness and permittivity). To facilitate this process, a CSV file is created that contains the filenames of the B-scan data alongside their corresponding labels. Each entry in the CSV file will have the filename, wall thickness, and permittivity values.

Once all the necessary functions are defined, the next step is to execute the entire simulation pipeline, which involves iterating through the list of wall parameters to generate the B-scan datasets, extract the E_Z components, create the corresponding labels, and compile the data and labels into a CSV file.

```
for param in parameters:
            thickness, permittivity = result
            input_file, output_file, merged_filename = create_input_file(thickness, permittivity, index)
            run_simulation(input_file, n_iterations)
            merge_output_files(output_file)
            ez_file_path = move_merged_file(output_file, index)
            data.append({
                'index': index,
                'filename': ez_file_path,
                'thickness': thickness,
                'permittivity': permittivity
            })
```

Figure 14

The resulting CSV file will contain the complete training data, which consists of the B-scan data filenames and their corresponding wall parameters (thickness and permittivity):

index	filename	thickness	permittivity
1	simulation_1bscan_Ez.npy	0.15	2.0
2	simulation_2bscan_Ez.npy	0.175	2.0
3	simulation_3bscan_Ez.npy	0.2	2.0
4	simulation_4bscan_Ez.npy	0.225	2.0
5	simulation_5bscan_Ez.npy	0.25	2.0
6	simulation_6bscan_Ez.npy	0.275	2.0
7	simulation_7bscan_Ez.npy	0.3	2.0
8	simulation_8bscan_Ez.npy	0.325	2.0
9	simulation_9bscan_Ez.npy	0.35	2.0
10	simulation_10bscan_Ez.npy	0.375	2.0
11	simulation_11bscan_Ez.npy	0.4	2.0
12	simulation_12bscan_Ez.npy	0.425	2.0
13	simulation_13bscan_Ez.npy	0.45	2.0
14	simulation_14bscan_Ez.npy	0.475	2.0
15	simulation_15bscan_Ez.npy	0.5	2.0
16	simulation_16bscan_Ez.npy	0.15	2.25
17	simulation_17bscan_Ez.npy	0.175	2.25
18	simulation_18bscan_Ez.npy	0.2	2.25
19	simulation_19bscan_Ez.npy	0.225	2.25
20	simulation_20bscan_Ez.npy	0.25	2.25

Figure 15

The next step involves data preparation for CNN model training. The CSV file containing the filenames of the B-scan data and their parameters is read into a data frame using Pandas framework. For each row of the data frame, the B-scan filename, and its associated wall parameters, such as thickness and permittivity, are extracted. The B-scan data corresponding to its filename will be loaded as a NumPy array and appended to a list of training B-scan data called *train_bscan* array. The associated wall parameters are collected into a separate array called *train_labels*. The labels are also converted into a NumPy array.

```
for index, row in labels_df.iterrows():
    filename = row['filename']
    labels = row[['thickness', 'permittivity']].values
    image_path = os.path.join(bscan_folder, filename)
    if os.path.exists(image_path):
        image = np.load(image_path)
        train_bscan.append(image)
        train_labels.append(labels)

train_labels = np.array(train_labels, dtype=np.float32)
train_bscan = np.array(train_bscan, dtype=np.float32)
```

Figure 16

Each B-scan dataset consists of 2D numerical data, as it is a collection of multiple A-scan data. The dimensions of the B-scan data are determined by the number of iteration (*n*) times the size of each A-scan data, which consists of 1485 data points. In this case, each B-scan data has dimensions of *1485 x 15*. With a total of 255 B-scan samples, the resulting *train_bscan* array has dimensions of *255 x 1485 x 15*.

To properly format the B-scan data for CNN training, an additional dimension is required to match the channel input expected by CNNs. Typically, CNNs are designed to process 3 channels for RGB images, but in this case, the B-scan data contains solely of intensity values, representing the E_Z component. Since the consists only intensity values, a single channel is added. As a result, the final shape of the *train_bscan* array becomes *255 x 1485 x 15 x 1*, where the '1' represents the intensity channel.

Each label consists of two values representing the wall parameters, formatted as [*wall thickness, wall permittivity*]. Therefore, the *train_labels* will have dimensions of *255 x 2*. Both the *train_bscan* and *train_labels* arrays will be used in the next chapter for training the CNN models to estimate the wall parameters.

Below are a few examples of randomly data selected from *train_bscan* plotted with GPRMax's tools along with their corresponding parameters in *train_labels.*

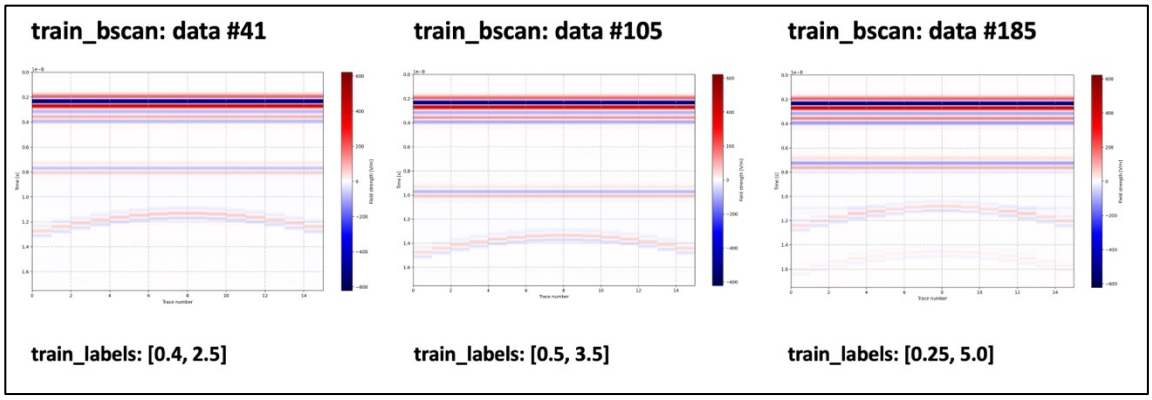


Figure 17

4.2. Generating B-Scan Dataset for Different Object Positions (Second Model)

In this section, another set of datasets will be generated, following a similar setup to the previous simulation. However, in this simulation, the object position will be varied. This variation will add another dimension to the data, which might allow the model to learn how different object positions affect the B-scan data. By systematically altering the object's coordinate, a more diverse dataset can be created.

The second simulation will have four parameters, which are, wall thickness, wall permittivity, and the x and y position of the object. The wall thickness varies from 0.15 m to 0.5 m with a step size of 0.05 m. The wall permittivity ranges between values in [2,6], with increments of 0.5. For the object, the position will start from the top left. It will be moved 0.35 m each step in both horizontal and vertical directions. The object has x-coordinate in (0.1 + radius, 1.9 – radius) meters range, and y-coordinates in (0.1 + radius, 1.6 – wall thickness – radius) meters range. The configuration for this simulation is shown in the image:

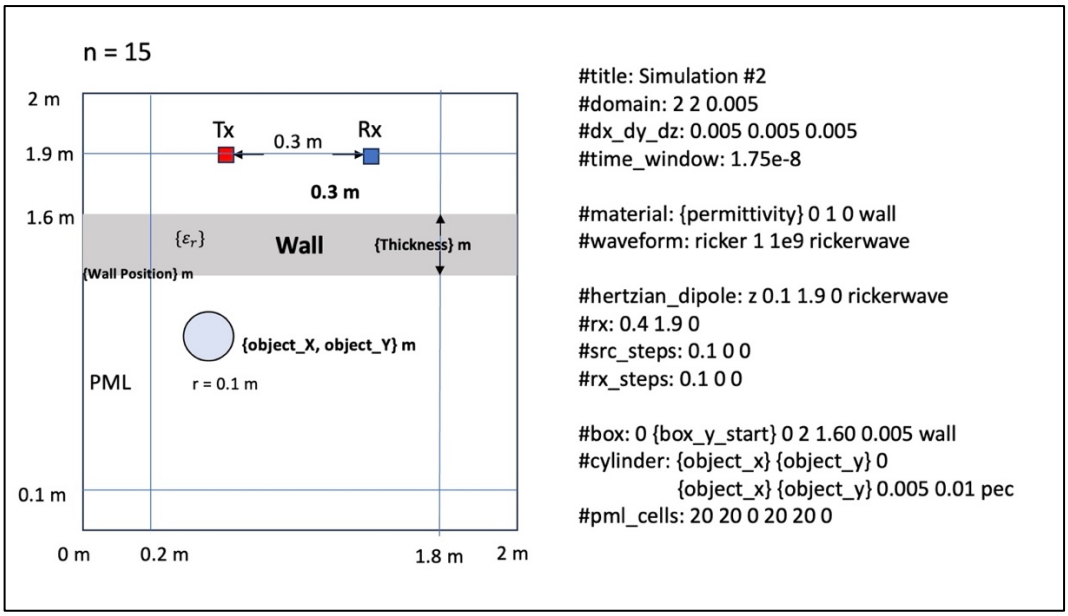


Figure 18

Next, a list of 1215 unique combination of simulation parameters will be created. The same process will be followed to generate input scripts, run simulations, merge the resulting output files, extract the E_Z data and create NumPy arrays. The labels, which consist of four parameters will be generated for each data. Then, all the data and corresponding labels will be organized into a CSV file.

Wall_parameters2 = [(thickness1, permittivity1, object_x1, object_y1), (thickness2, permittivity2, object_x2, object_y2), ...(thickness1215, permittivity1215, object_x1215, object_y1215)]

Once the CSV file has been created, the next step is to prepare the dataset for training the CNN models. The CSV file will be read to extract the filenames of the E_Z data along with their parameters. Then it will be organized into two arrays, *train_bscan2* which contains the b-scan data, and *train_labels2* which contains the training parameters.

```
for index, row in labels_df.iterrows():
    filename = row['filename']
    labels = row[['thickness', 'permittivity', 'object_x', 'object_y']].values
    image_path = os.path.join(bscan_folder, filename)
    if os.path.exists(image_path):
        image = np.load(image_path)
        train_bscan.append(image)
        train_labels.append(labels)

train_labels2 = np.array(train_labels, dtype=np.float32)
train_bscan2 = np.array(train_bscan, dtype=np.float32)
```

Figure 19

The train_bscan array has dimensions of 1215 x 1485 x 15 x 1. It indicates 1215 samples of B-scan data, where each sample is a 2D array of size 1485 x 15 and includes a single channel to represent the intensity. The train_labels array has dimensions of 1215 x 4, meaning each data consists four values and formatted as [*wall thickness, wall permittivity, object x-coordinate, object y-coordinate*].

4.3. Generating B-Scan Dataset for Different Object Properties (Third Model)

In this section, the third dataset will be generated. The dataset will simulate scenarios with objects that have different permittivity and varying position. The objective is to investigate how changes in object's properties affect the B-scan data and model predictions. A comprehensive list of simulation parameters will be created, including various combinations of object permittivity and position.

For this third dataset, an additional parameter will be added, hence there will be five parameters in total: two wall parameters (thickness and permittivity) and three object attributes (permittivity, x-coordinate, and y-coordinate). The wall thickness is set from 0.15 m to 0.5 m, with steps of 0.05 m, and the wall permittivity is set from 2 to 6 with steps of 1. The object permittivity will be set between the values of 2, 4, and 6. The object's x and y coordinates will be adjusted at steps of 0.3 m in both the horizontal and vertical directions, following the same range as the previous simulation.

Next, a list containing a total of 2700 parameter variations will be created. The next processes will be the same as the previous simulations, which results in two arrays: *train_bscan3,* which contains the B-scan data, and *train_labels3*, which contains the corresponding labels. The *train_bscan3* array will have dimensions of 2700 x 1485 x 15 x 1, while the *train_labels3* array will have dimensions of 2700 x 5. Each label will be formatted as [*wall thickness, wall permittivity, object permittivity, object x-coordinate, object y-coordinate*].

Chapter 5 : CNN Model Development and Evaluation

In this chapter, the development and evaluation of a CNN model for estimating wall parameters will be discussed. The primary objective of the CNN model is to learn the spatial patterns in the two-dimensional B-scan data to estimate the wall parameters. Additionally, the model will also learn to estimate object parameters. This approach aims to effectively analyze the B-scan data using deep learning algorithms.

5.1. Model Summary and Architecture

Below is the code for constructing the CNN model:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.models import Sequential

model = Sequential([
    layers.Input(shape=(train_bscan.shape[1], train_bscan.shape[2], 1)),

    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='linear')
])
```

Figure 20

The CNN model is constructed using TensorFlow and Keras software, which are widely used in machine learning models. The model is designed to process single-channel B-scan images for predicting wall parameters, such as thickness and permittivity, as well as object parameters. The CNN architecture consists of several layers to extract spatial features and make accurate predictions.

Below is the summary of the CNN model architecture created with TensorFlow:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 1485, 15, 32)	320
max_pooling2d (MaxPooling2D)	(None, 742, 7, 32)	0
conv2d_1 (Conv2D)	(None, 742, 7, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 371, 3, 64)	0
conv2d_2 (Conv2D)	(None, 371, 3, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 185, 1, 128)	0
flatten (Flatten)	(None, 23680)	0
dense (Dense)	(None, 128)	3,031,168
dense_1 (Dense)	(None, 2)	258

Total params: 3,124,098 (11.92 MB)
Trainable params: 3,124,098 (11.92 MB)
Non-trainable params: 0 (0.00 B)

Figure 21

The model architecture consists of the following layers:

- **Input Layer:** This layer accepts B-scan images of shape (height, width, channels). In this project, the B-scan data only represents signal strength, so there is no RGB color. Therefore, the input size of the model will be (height, width, 1)

- **Convolutional Layers:** Three convolutional layers are used in this project. Behind each layer, a ReLU activation and max pooling are applied to reduce the output dimensions. Convolutional layers use filters to process the input, which will generate feature maps that highlight significant patterns in the data [4]. In this project, the numbers of filters used are 32 for first layer, 64 for second layer, and

128 for third layer. The first layer extracts lower-level feature, while the last layer extracts higher-level patterns.

- **Flatten Layer:** This layer is used to flatten the multi-dimensional data into one-dimensional vector. This result will become the input to the fully connected layer, which will make the estimation of wall parameters value.

- **Dense Layers**: Following the flatten layer, a fully connected layer with 128 neurons is used to combine extracted patterns and generate predictions. The final output layer is a fully connected layer, featuring a number of neurons which matches number of target parameters for prediction. A linear activation function is utilized at the output layer for regression tasks to predict the numerical values.

5.2. Model Training and Evaluation

In this section, the training process of the CNN model using three different datasets will be explained. Each dataset represents unique simulation parameters, and the model will be trained on all of them to predict wall and object properties.

5.2.1 Model Training and Evaluation on First Dataset: Predicting Wall Parameters

The first dataset consists of 255 B-scan data generated from simulations that vary only in wall thickness and wall permittivity (as explained in **section 4.1**). The dataset structure is:

B-Scan Data (*train_bscan*): (*255, 1485, 15, 1*)
B-Scan Labels (*train_labels*): (*255, 2*)

The CNN model is trained to learn the relationship between the B-scan and the two wall parameters. To ensure a robust evaluation on the model, the B-scan dataset is separated into three different parts: training set, validation set and test set. The train and validation dataset consist of 90% of total data, while the remaining data will be used for testing.

Specifically:

- **Train Dataset**: This consists 72% of the B-scan data or **183** samples. This is used for learning the relationships between the training data and the labels.

- **Validation Dataset**: This consists 18% of the B-scan data or **46** samples. This is used to monitor the model's performance during training and helps in detecting overfitting.

- **Test Dataset**: This consists of the remaining 10% of unused B-scan data or **26** samples. This data will not be used in the training phase. It will be reserved for the model's performance evaluation and prediction.

The *train_test_split* from the *sklearn.model_selection* module is used to divide the dataset.

```
X_temp, X_test, y_temp, y_test = train_test_split(train_bscan, train_labels, test_size=0.1)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2)
```

Figure 22

In the CNN model for this simulation, the final dense layer will consist of two neurons, where it will predict the wall thickness and permittivity separately.

```
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(2, activation='linear')
```

Figure 23

The model uses the Adam optimizer with a learning rate of 0.001. Since this is a regression task, the Mean Squared Error (MSE) will be used as the loss function. The MSE is used to determine the loss value, which is the average squared difference between the predicted and actual values. The goal of the training process is to minimize this loss value. The model is trained over 50 epochs, and the network's weights will be updated each time to reduce the loss.

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='mean_squared_error',
              metrics=['mae'])
```

Figure 24

The CNN model is trained using the *fit* function, using the training and validation data as inputs. The model runs for over 50 epochs, with a batch size of 64 samples per iteration. The validation data helps monitor the model's performance during the training process by providing feedback on how well it generalizes to different set of data.

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=64
)
```

Figure 25

To visualize the model's performance during training, the loss function is plotted for both the training and validation datasets across all epochs. This is to monitor how well the model is learning. The graph is generated using the loss values stored in the *history* object.

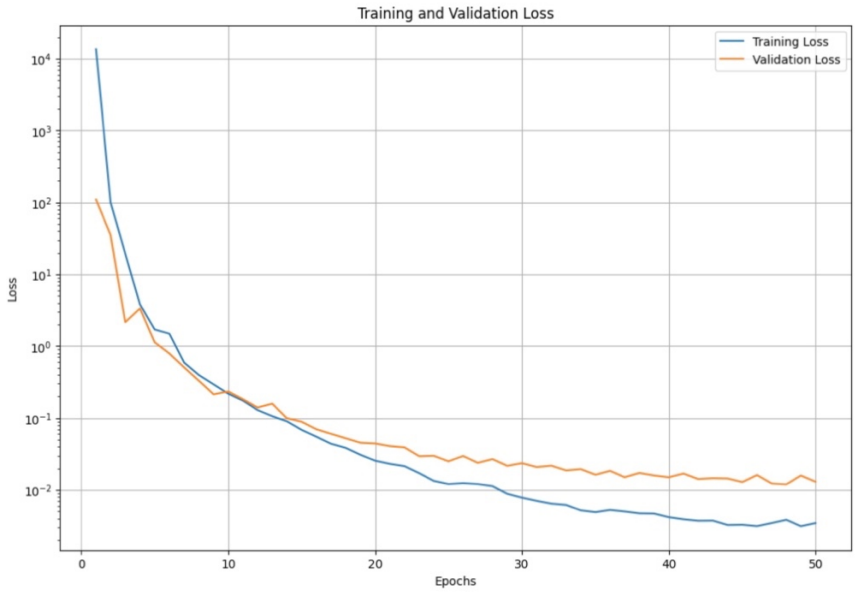


Figure 26

The final loss values are 0.0038 for training and 0.013 for validation. The mean absolute error (MAE) is 0.0442 for training and 0.0716 for validation.

The evaluation of the CNN model on test dataset is performed using the *evaluate* function, which results in two important metrics: the loss (calculated using the MSE) and the mean absolute error (MAE). The MAE represents the average of the absolute differences between predicted and actual values. These metrics are used to assess the model's performance on the test dataset. Additionally, the *predict* function is employed to generate predictions on the test dataset.

```
loss, mae = model.evaluate(X_test, y_test)
predictions = model.predict(X_test)
```

Figure 27

The loss value for the test dataset is 0.0136, and the mean absolute error (MAE) is 0.0843. The table below summarizes the prediction results of the CNN model on all 26 test data points, comparing them with the actual simulation parameters.

Test Sample	Actual Wall Thickness (m)	Predicted Wall Thickness (m)	Actual Wall Permittivity	Predicted Wall Permittivity
1	0.4	0.373	5	4.858
2	0.3	0.223	2	1.729
3	0.25	0.277	3.25	3.311
4	0.4	0.36	5.25	5.21
5	0.45	0.453	3.75	3.862
6	0.275	0.259	5	5.115
7	0.3	0.207	5.25	5.164
8	0.2	0.222	4.75	4.778
9	0.375	0.28	2	1.888
10	0.15	0.109	2.5	2.468
11	0.15	0.11	5	4.909
12	0.425	0.426	5.5	5.173
13	0.25	0.145	5.75	5.498
14	0.3	0.215	5.5	5.285
15	0.175	0.186	4.25	4.443
16	0.35	0.375	3	3.069
17	0.225	0.214	5.25	5.111
18	0.15	0.074	2.25	2.235
19	0.3	0.355	3.5	3.584
20	0.375	0.359	2.25	2.23

21	0.375	0.337	5.75	5.407
22	0.25	0.185	2.25	2.1
23	0.15	0.109	4	3.948
24	0.2	0.216	4.5	4.699
25	0.225	0.188	2.5	2.488
26	0.250	0.224	4	4.13

To further analyze the model's prediction, histograms are created to visualize the distribution of the differences between the predicted values and the actual values for each parameter: the wall thickness and permittivity. The two histograms show the frequency of prediction errors and provide insights into the model's performance in each parameter.

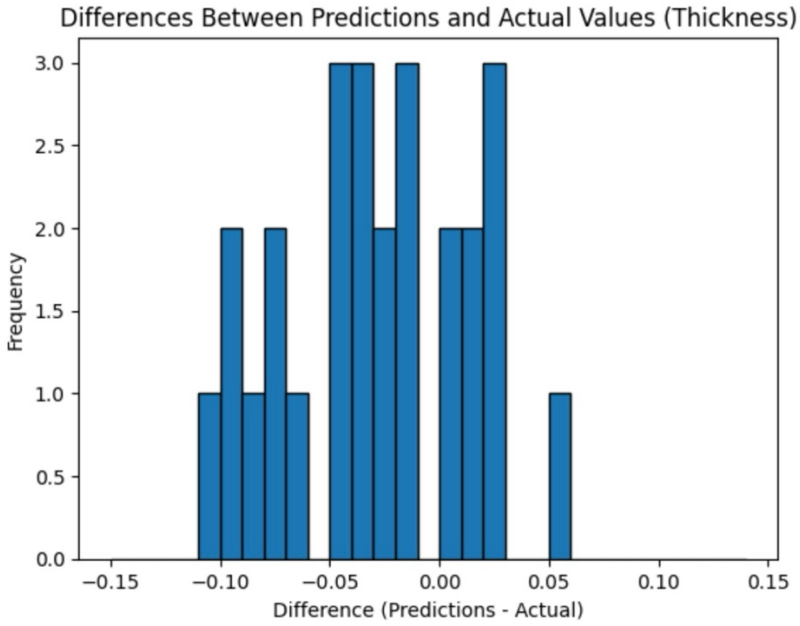


Figure 28

For the wall thickness, the range of the prediction errors *(predicted – actual)* spans from -0.105 m to 0.055 m. The calculated mean absolute error (MAE) for the wall thickness is 0.0419 m. This result indicates that on average, the CNN model mispredicted the wall thickness by approximately 0.04 m. Considering that the range of wall thickness in this simulation is between 0.15 m and 0.5 m, the prediction errors are still considered acceptable.

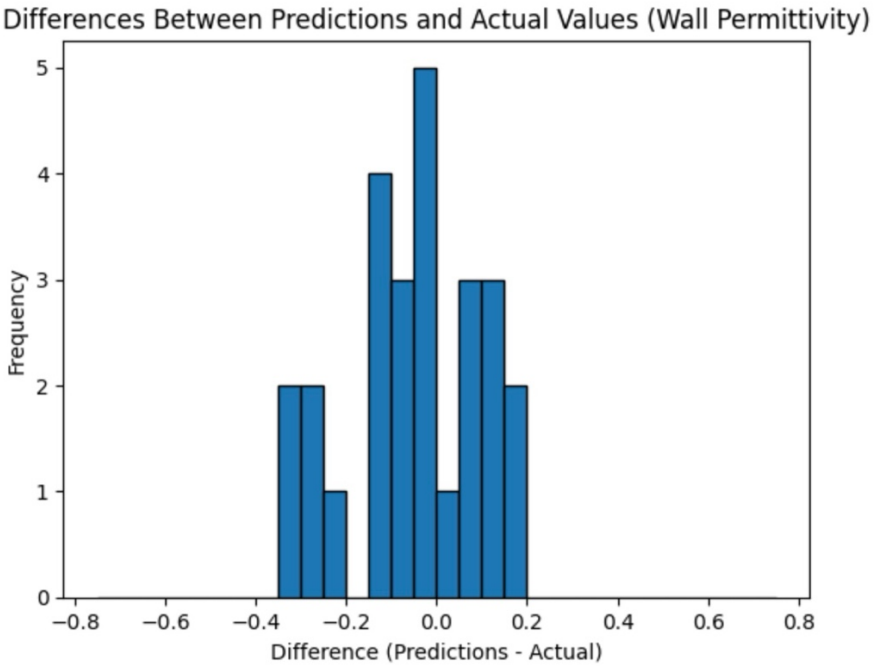


Figure 29

In terms of the wall permittivity, the range of value differences *(predicted – actual)* is from -0.343 to 0.199. The mean absolute error (MAE) for the wall permittivity is 0.1265. Notably, more than 80% of the predicted results fall within a 0.2 difference from the actual values, which reflects a strong performance by the model in this task. Considering that the range of wall permittivity in the simulation varies from 2 to 6, the model's predictions demonstrate a good level of accuracy.

5.2.2 Model Training on Second Dataset: Predicting Wall Parameters and Object Position

In this section, the CNN model will be trained on the second dataset (as described in **section 4.2**). In this dataset, there are four parameters to be predicted: wall thickness, wall permittivity, object x-coordinate, and object y-coordinate. The dataset consists of:

B-Scan Data #2 (*train_bscan2*): *1215 x 1485 x 15 x 1*
B-Scan Parameters #2 (*train_labels2*): *1215 x 4*

Following the same steps as in the previous model training, the dataset will be divided into three parts using the *train_test_split* method:

- **Train Dataset**: This consists 72% of the B-scan data or **875** samples.
- **Validation Dataset**: This comprises 18% of the B-scan data or **219** samples.
- **Test Dataset**: This includes the remaining 10% of the B-scan data or **121** samples.

For this model, the final dense layer will be modified to output four values, since the aim is to predict four parameters in this case.

Like the first model, this model is also trained with Adam optimizer and a learning rate of 0.001. The model's loss function is the mean squared error (MSE). Training is performed using the *fit* function with 50 epochs and a batch size of 64. The figure below illustrates the training and validation loss throughout the training process over 50 epochs.

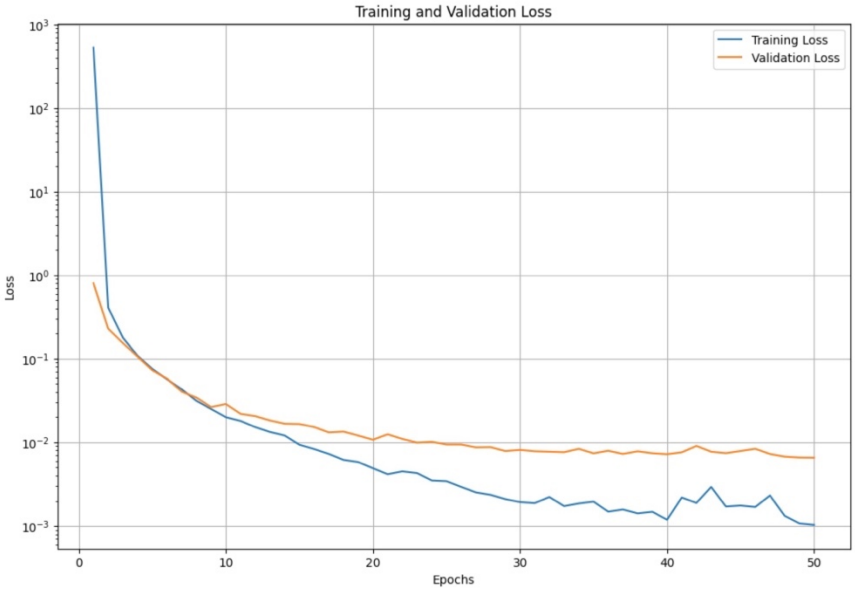


Figure 30

The loss values for the second model are 0.00096 for the training dataset and 0.0066 for the validation dataset. The mean absolute errors (MAE) are 0.0228 for training and 0.0565 for validation. By using the *evaluate* function, the loss value and MAE for all parameters

on the test dataset can be calculated as 0.009 and 0.0621, respectively. The figures below show the histograms that illustrate the distribution of prediction differences for the simulation parameters.

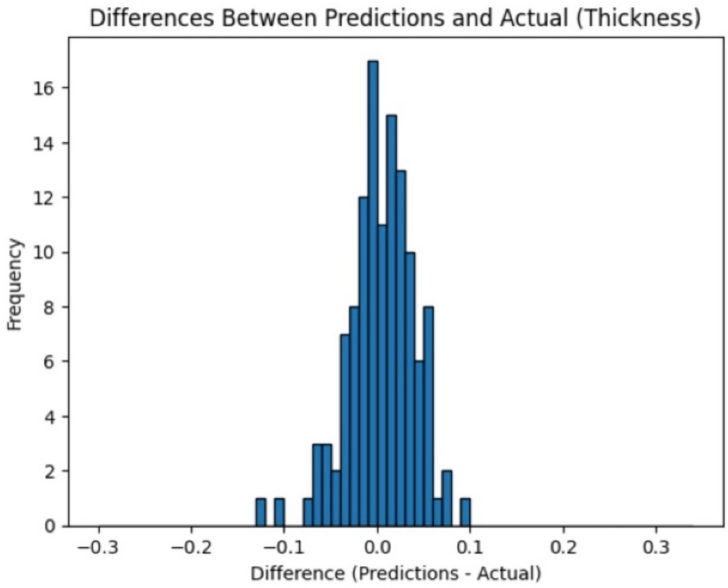


Figure 31

The prediction errors for the wall thickness range between -0.125 m and +0.09 m with most errors (82%) fall within -0.05 m and +0.05 m. The calculated mean absolute error for the wall thickness is 0.028 m.

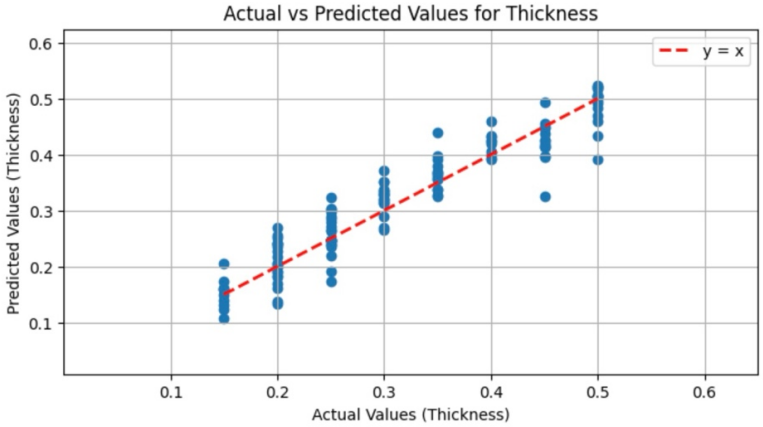


Figure 32

The scatter plot above indicates a strong correlation between the predicted and actual wall thickness values. While there are some errors, the predictions are generally quite close with the simulation values, with most deviations not larger than 0.1 m.

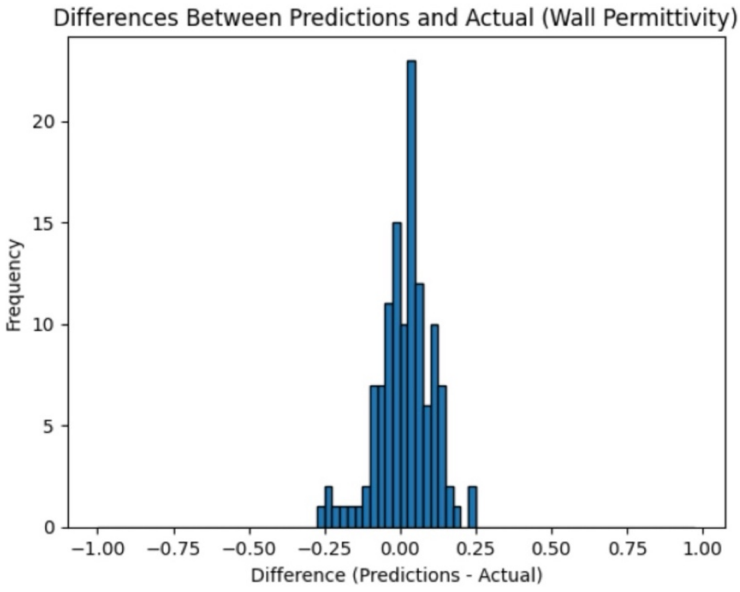


Figure 33

For the wall permittivity, the prediction errors range from -0.26 to 0.23, with approximately 90% of the errors falling within the interval of -0.15 to 0.15. The mean absolute error for wall permittivity is calculated to be 0.069.

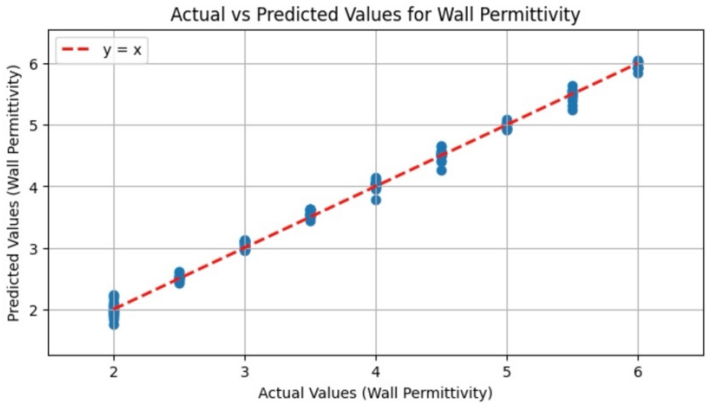
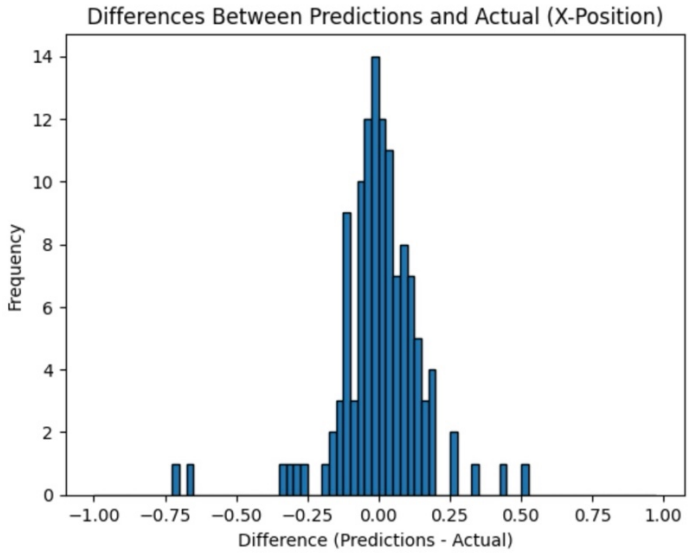
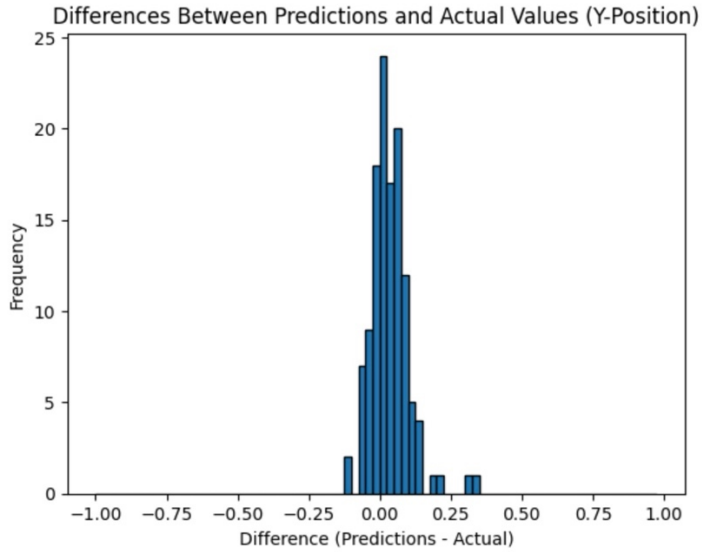


Figure 34

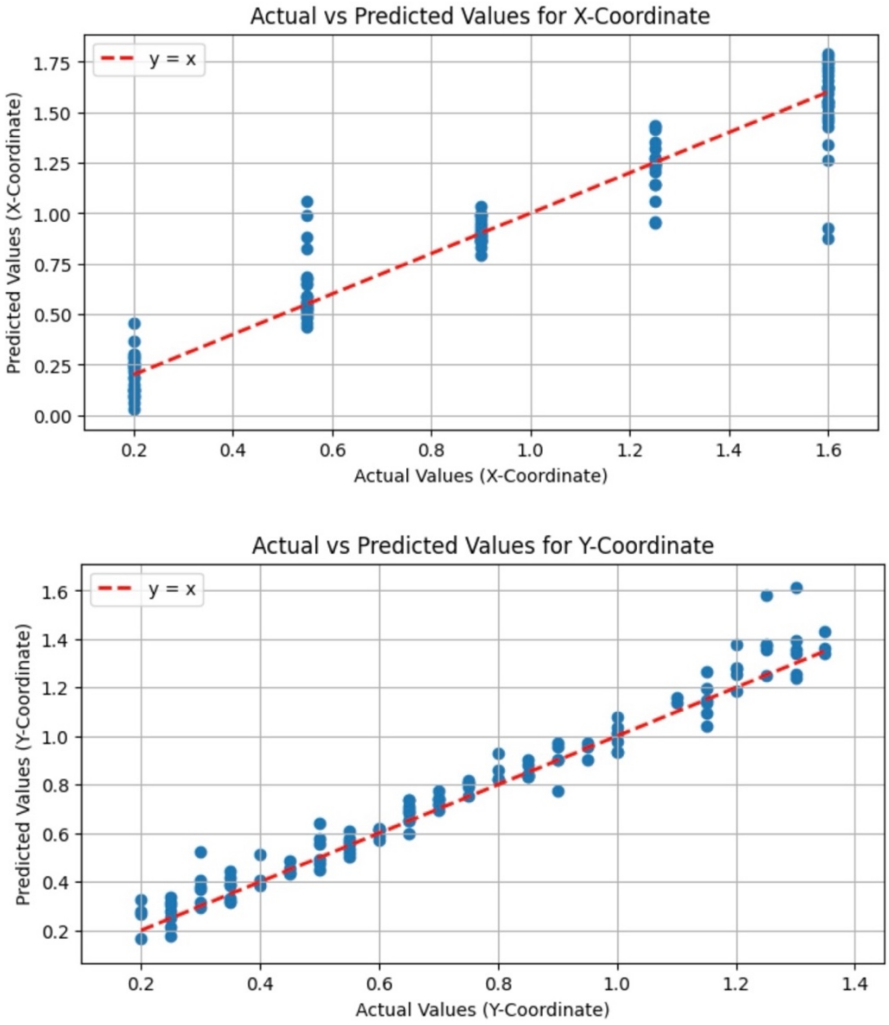
The scatter plot for the wall permittivity shows a high level of accuracy in the model's prediction, with only small deviations in the predicted values.





Figures 35 & 36

For the position of the object, the prediction error for X-coordinate range from -0.723 m to 0.513 m, while for Y-coordinate, the range is from -0.123 m to 0.329 m. The mean absolute error (MAE) is 0.1 m for X-coordinate and 0.053 m for Y-coordinate.



Figures 37 & 38

The scatter plots show that X-coordinate predictions have more errors than Y-coordinate predictions. This is likely due to the low resolution of the B-scan data, which only includes 15 A-scans. With fewer scans, the model has less lateral information, which makes it harder to accurately predict the X-positions, especially near the edges. In contrast, Y-coordinate predictions depend more on the timing of the reflected signals, which is more consistent and less impacted by the resolution. The model also performs better at predicting X-positions near the centre, while error increases near the edges due to noisier data.

5.2.3 Model Training on Third Dataset: Predicting Wall Parameters, Object Position and Object Permittivity

In this section, the model will be trained on a more complex dataset. The third dataset (as described in **section 4.3**) has five parameters. This structure of the dataset consists of:

B-Scan Data #3 (*train_bscan3*): *2700 x 1485 x 15 x 1*
B-Scan Parameters #3 (*train_labels3*): *2700 x 5 [wall thickness, wall permittivity, object X-co ordinate, object Y-coordinate, and object permittivity]*

The data will be divided into three parts:

- **Train Dataset**: This consists 72% of the B-scan data or **1944** samples.
- **Validation Dataset**: This consists 18% of the B-scan data or **486** samples.
- **Test Dataset**: This consists 10% of the B-scan data or **270** samples.

The model architecture and training steps will remain consistent with the previous models. However, the final dense layer will be modified to five units to accommodate the five labels present in the model. The same hyperparameters, including the number of epochs, batch size, and loss function, will also be retained for this model. The plot of training and validation loss for this model are shown below.

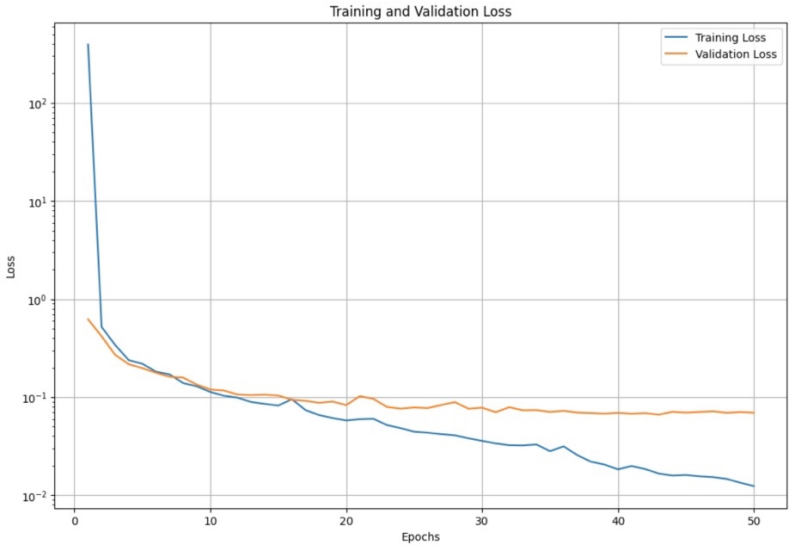
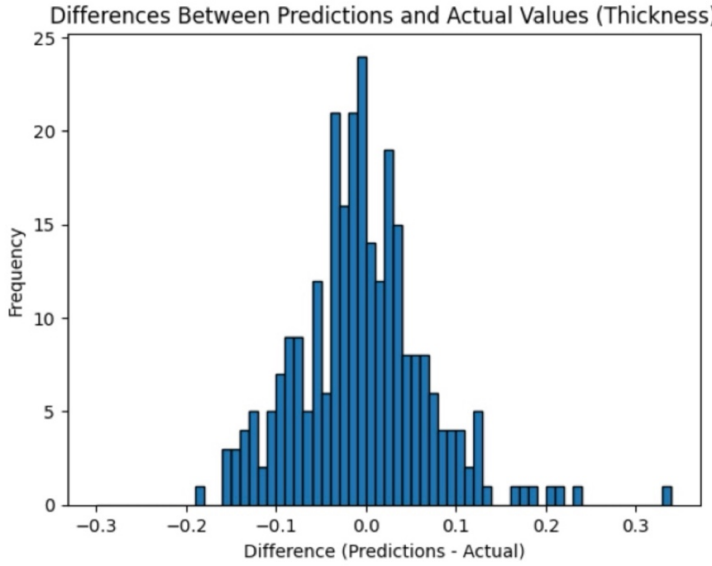


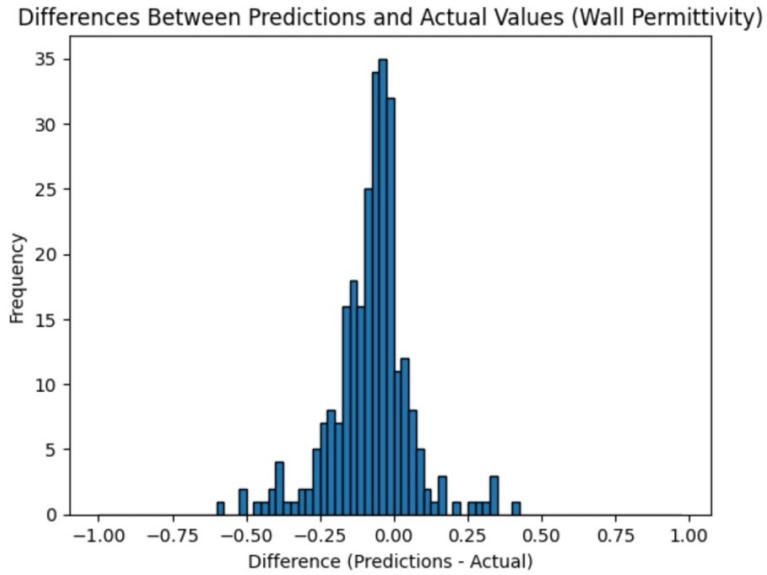
Figure 39

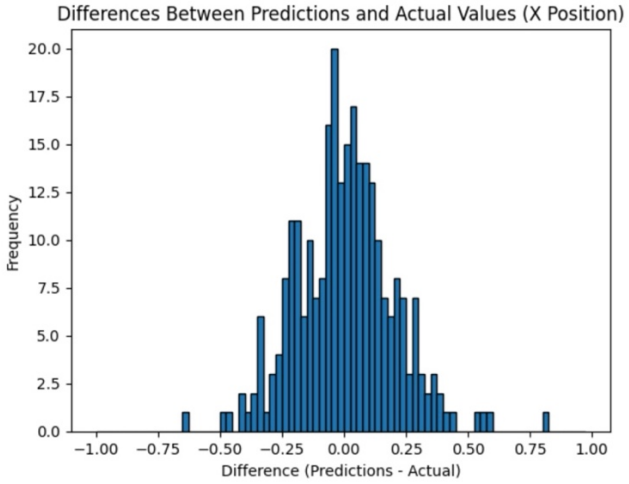
The table below summarizes the loss values and mean absolute errors (MAE) during training, validation, and testing.

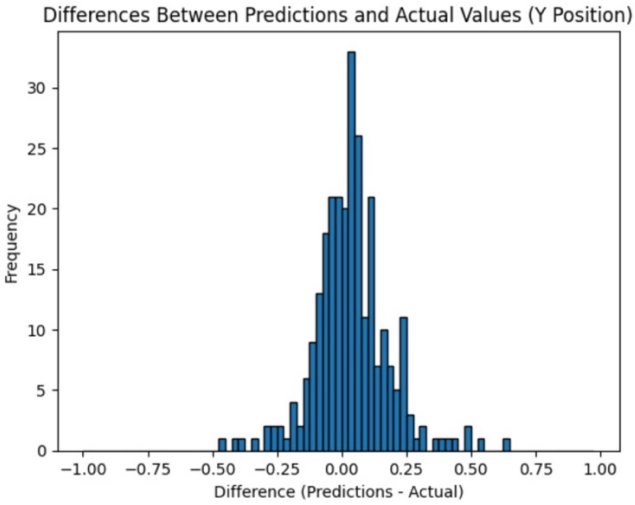
	Train Data	Validation Data	Test Data
Loss Value	0.0128	0.0695	0.0805
MAE	0.0700	0.1612	0.1656

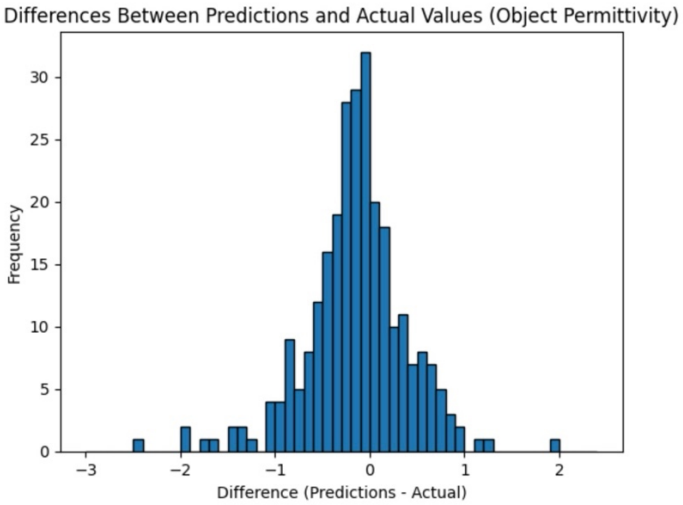
The figures below show five histograms which shows the prediction errors for wall parameters: thickness and permittivity, position, and permittivity of the object.






Differences Between Predictions and Actual Values (X Position)
Frequency
Difference (Predictions - Actual)


Differences Between Predictions and Actual Values (Y Position)
Frequency
Difference (Predictions - Actual)


Differences Between Predictions and Actual Values (Object Permittivity)
Frequency
Difference (Predictions - Actual)

Figures 40, 41, 42, 43, 44

The summary of the prediction result for the third model is shown in the table below.

Parameters (Range)	Minimum Prediction Error	Maximum Prediction Error	Mean Absolute Error (MAE)
Wall Thickness (0.15 m – 0.55 m)	-0.188 m	+0.336 m	0.054 m
Wall Permittivity (2 – 6)	-0.59	+0.419	0.113
Object Permittivity (2 – 6)	-2.472	+1.915	0.399
Object X-Coordinate (0.2 m – 1.7 m)	-0.637 m	+1.017 m	0.155 m
Object Y-Coordinate (0.2 m – 1.35 m)	-0.462 m	+0.647 m	0.107 m

The largest prediction error occurs in object permittivity, with a value of 0.399. The errors for the other predictions are smaller, with margins of 0.054 m for wall thickness, 0.113 for wall permittivity, 0.155 m for the object's X-coordinate, and 0.107 m for the object's Y-coordinate. The higher error in object permittivity, compared to the wall, may be influenced by the smaller surface area, which provide less information for the model.

The model is expected to predict values within the error range defined by the mean absolute error (MAE). This range indicates the expected differences from the true values. For example, if a B-scan data label has a label of [0.2, 3, 4, 1.1, 0.6], the predicted values are expected to be between [0.146, 2.887, 3.601, 0.945, 0.493] and [0.254, 3.113, 4.399, 1.255, 0.707].

Chapter 6 Conclusion and Future Work

5.1 Conclusion

This project developed Convolutional Neural Network (CNN) models to estimate wall and object parameters for Through-the-Wall Imaging (TWI) simulations using GPRMax-generated B-scan data. The model was tested on three datasets, each with different complexity.

In the first dataset, which only wall parameters (thickness and permittivity) are considered, the model accurately predicted both thickness and permittivity with an average error of 0.08. This is a good result considering the small dataset size of 255. In the second dataset, which introduced object positions (X-coordinate, Y-coordinate), the model become more complex. While it maintained good accuracy for the wall parameters, slight errors were observed in the object positions estimation, especially for the X-coordinate. However, this model still performed well, with a mean absolute error (MAE) if 0.06, the lowest among the three models. The third dataset was the most complex of all, with five parameters to predict. The model's MAE increase to 0.17, with the largest drop in accuracy for object related parameters, especially permittivity.

Overall, the models performed well in estimating wall parameters, though the overall accuracy decreased with the addition of more complex object-related predictions such as object position and permittivity.

5.2 Recommendation in Future Work

For future work, several improvements can be done to improve the current model's performance. First, the dataset can be expanded by increasing both the size and diversity of the parameters, such as wall thickness, permittivity, and object properties. Additionally, future simulations could incorporate more complex wall structures, such as multi-layered

walls, instead of the homogeneous wall. Another area of improvement is to expand the model to handle multiple objects behind the wall. In this project, only a single object is considered, but real world scenarios often involve dealing with multiple objects with different properties at the same time. Introducing more variety in the data would help the model learn to recognize multiple reflections in the B-scan, which could help to make more accurate predictions.

Reflection on Learning Outcome Attainment

During my FYP, I focused on machine learning and synthetic radar image generation, which allowed me to apply key concepts from electrical engineering and programming to solve a real-world problem.

Firstly, my engineering knowledge was essential throughout the project. Generating B-scan data for the simulation required a deep understanding of engineering electromagnetics, which helped me to create reliable dataset for training the Convolutional Neural Network (CNN). Familiarizing myself with CNNs was also crucial, as this project heavily relied on deep learning techniques to extract the dataset features and make prediction of wall parameters, which could enhance the TWI simulation and object detection.

Additionally, problem analysis was critical. Generating the dataset and building a CNN model required an analytical approach, as I needed to understand the key parameters and variables which could impact the TWI simulation. I also had to evaluate which parameters the model could accurately predict to ensure it could provide reliable insights.

Finally, I relied heavily on modern tools, such as GPRMax software for simulations and TensorFlow to streamline the machine learning workflow. Using the right tools made the project more efficient. Overall, my FYP experience has improved my engineering skills, problem analysis abilities, and familiarity with modern tools, all of which will be useful in the future.

Acknowledging/Declaring the Use of GAI

Please refer to NTU's Current Policy & Guidelines on the Use of Generative AI available in NTUlearn home page and the link: https://entuedu.sharepoint.com/sites/Student/dept/ctlp/SitePages/Exploring-the-Impact-of-Generative-Artificial-Intelligence-(GAI)-Tools-on-Education.aspx

1. Complete the following declaration if applicable.
2. Create a Paper Trail to document the input prompt, output obtained, and how you have used it.

I, Christian Joseph, cjoseph001@e.ntu.edu.sg honestly and sincerely make the following declaration in relation to the following course submission.

1. Name of course: Final Year Project
2. Course Code: EE4080
3. Instructor: Assistant Professor Abdulkadir C. Yucel
4. Title of Assignment/Project Submission: Deep-Learning Based Estimation of Wall Parameters for Through-the-Wall Imaging

In relation to the foregoing, I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

i. Used GAI as permitted to assist in generating key ideas only	☐
ii. Used GAI as permitted to assist in generating a first text only. And/or	☐
iii. Used GAI to refine syntax and grammar for correct language submission only. Or	☑
iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project.	☐

I also declare that I have:

a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
c. Apart from the foregoing notices, the submission is wholly my own work.

Christian Joseph
Student Name & Signature

Thursday, 7 Nov 2024
Date

References

[1] C. Warren, A. Giannopoulos, and I. Giannakis, “gprMax: Open source software to simulate electromagnetic wave propagation for Ground Penetrating Radar,” *Computer Physics Communications*, vol.209, pp. 163-170, December 2016.

[2] M. Abadi *et al*, “TensorFlow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16),* Savannah, GA, USA, pp.265-283, November 2016.

[3] P. Protiva, J. Mrkvica, and J. Machac, “Estimation of wall parameters from time-delay-only through-wall radar measurements,” *IEEE Transactions on Antennas and Propagation*, vol. 59, no. 11, pp. 4268–4278, November 2011.

[4] X. Liang, "Theoretical basis," in *Ascend AI Processor Architecture and Programming: Principles and Applications of CANN*, Elsevier, 2020, pp. 1–40.