THE DEEP LEARNING TECHNIQUES APPLIED TO ELECTROMAGNETIC IMAGING VIA GROUND-PENETRATING RADAR

DAI QIQI

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

06-12-2022

. . . . . .

Date

. . . . . .

Dai Qiqi

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

15-12-2022

. . . . . .

Date

. . . . . .

Asst. Prof. Abdulkadir C. Yucel

Authorship Attribution Statement

This thesis contains material from three papers published in the following peer-reviewed journals / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as Q. Dai, Y. H. Lee, H. H. Sun, G. Ow, M. L. M. Yusof, and A. C. Yucel, "DMRF-UNet: A two-stage deep learning scheme for GPR data inversion under heterogeneous soil conditions," *IEEE Transactions on Antennas and Propagation*, vol. 70, no. 8, pp. 6313-6328, 2022. The contributions of the co-authors are as follows:

- I designed the networks, performed the experiments and result analysis, and prepared the manuscript.
- Prof. Yucel provided the initial research idea, supervised the project, and revised the manuscript.
- Prof. Lee co-supervised the project and revised the manuscript.
- Dr. Sun co-performed the experiments and revised the manuscript.
- Mr. Yusof provided support for measurement data collection.
- Dr. Ow provided support for measurement data collection.

Chapter 4 is published as Q. Dai, Y. H. Lee, H. H. Sun, G. Ow, M. L. M. Yusof, and A. C. Yucel, "3DInvNet: A Deep Learning-Based 3D Ground-Penetrating Radar Data Inversion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1-16, 2023. The contributions of the co-authors are as follows:

- I designed the networks, performed the experiments and result analysis, and prepared the manuscript.
- Prof. Yucel provided the initial idea, supervised the project, and revised the manuscript.
- Prof. Lee co-supervised the project and revised the manuscript.
- Dr Sun co-performed the experiments and revised the manuscript.
- Mr. Yusof provided support for measurement data collection.
- Dr. Ow provided support for measurement data collection.

Chapter 5 is published as Q. Dai, Y. H. Lee, H. H. Sun, J. Qian, G. Ow, M. L. M. Yusof, and A. C. Yucel, "A deep learning-based GPR forward solver for predicting B-scans of subsurface objects," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022. The contributions of the co-authors are as follows:

- I established the methodology, performed the experiments and result analysis, and prepared the manuscript.
- Prof. Yucel proposed the initial research idea, supervised the project, and revised the manuscript.
- Prof. Lee co-supervised the project and revised the manuscript.
- Dr Sun assisted in the result analysis and revised the manuscript.
- Mr. Yusof provided support and revised the manuscript.
- Dr Ow provided support and revised the manuscript.

08-08-2023

. . . . . .

Date

Dai Qiqi

Acknowledgments

First of all, I would like to express my greatest gratitude to my main supervisor Assistant Professor Abdulkadir C. Yucel for his invaluable advice and continuous support throughout my PhD study. He introduced me into the world of research and taught me how to start and explore a research work. The discussions and meetings with him these four years helped me think deeper about my research. His passion and creativity for research encouraged and inspired me a lot.

I would also like to thank my co-supervisor Associate Professor Lee Yee Hui for her patient and gentle guidance. The financial support from Prof Lee's project is highly important for my PhD study and daily life. She provided significant and useful suggestions for my research, which helped me find the proper research direction. I learnt a lot from her rigorous attitude towards research.

Furthermore, it is my pleasure to thank Dr Hai-Hai Sun for her advice and patient assistance on my research and study. I am also deeply grateful to my colleagues and friends, Luo Wenhao, Ma Ling, Prabhu Shankar Mahendran, Jia Xiaofan, Wang Mingyu, Qian Jiwei, and Cheng Kaixuan. I would also like to express thanks to my friends and roommates, Yu Xiaoxi, Li Jiawei, Huo Lili, Liu Peng, Sun Zhongda, Wang Di, Zhou Siyuan, and Ai Xuan, with them I spend many great times.

I would also like to give my sincere gratitude to my parents. Their unconditional love and support made me relax when I'm tired and stressed. Finally, special thanks to my husband, Tao Ruijie, who accompanied me through these years and encouraged me to face problems bravely and solve them calmly. Your patience and love make this PhD journey more enjoyable.

Contents

List of Figures

List of Tables

Abstract

Ground-penetrating radar (GPR) has been widely used for non-destructive inspection of subsurface structures. In GPR inverse problem, the recorded GPR data is used to reconstruct subsurface permittivity maps, of great significance for mapping the subsurface utilities as well as detecting, recognizing, and monitoring the subsurface objects and living entities (such as tree roots). Traditional iterative algorithms developed for reconstructing permittivity maps from GPR data suffer from high computational costs and low accuracy when applied to complex scenarios. To tackle these issues, deep learning-based techniques have recently been proposed to characterize the mapping from the GPR data to the subsurface permittivity maps. However, these deep learning techniques suffer from three limitations. First, these techniques only take into account the ideal homogeneous soil environments; the noise and clutter due to heterogeneity of the subsurface environment are neglected, which can severely interfere with the objects' signatures and lower the accuracy of reconstructed permittivity maps. Second, since these techniques only reconstruct the 2D (sectional) permittivity maps from B-scans, they cannot provide 3D geometrical information about the subsurface objects, such as their complete shapes and orientations. Third, these techniques use traditional physics-based GPR forward solvers for dataset generation, which require excessive computational time, especially when a large dataset is required for training and testing stages. To address these limitations of deep learning algorithms for GPR data inversion and imaging, three deep learning-based approaches for reconstructing 2D/3D subsurface permittivity maps and generating B-scans for given subsurface scenarios are proposed in this thesis.

First, a two-stage deep neural network (DNN), called DMRF-UNet, is proposed to reconstruct the permittivity distributions of subsurface objects from GPR B-scans under heterogeneous soil conditions. In the first stage, a U-shape DNN with multi-receptive-field convolutions (MRF-UNet1) is built to remove the clutter due to the inhomogeneity of the heterogeneous soil. Then the denoised B-scan from the MRF-UNet1 is combined with the noisy B-scan to be inputted to the DNN in the second stage (MRF-UNet2). The MRF-UNet2 learns the inverse mapping relationship and

reconstructs the permittivity distribution of subsurface objects. To avoid information loss, an end-to-end training method combining the loss functions of two stages is introduced. A wide range of subsurface heterogeneous scenarios and B-scans are generated to evaluate the inversion performance. The tests performed on numerical and real measurement data show that the proposed network reconstructs the permittivities, shapes, sizes, and locations of subsurface objects with high accuracy. The comparison of the proposed DMRF-UNet with existing methods demonstrates its superiority for the inversion under heterogeneous soil conditions.

Next, a 3D deep learning scheme, called 3DInvNet, is proposed to reconstruct 3D permittivity maps from GPR C-scans. The proposed scheme leverages a prior 3D CNN with a feature attention mechanism to suppress the noise in the C-scans due to subsurface heterogeneous soil environments. Then a 3D U-shaped encoder-decoder network with multi-scale feature aggregation modules is designed to establish the optimal inverse mapping from the denoised C-scans to 3D permittivity maps. Furthermore, a three-step separate learning strategy is employed to pre-train and fine-tune the networks. The proposed scheme is applied to numerical simulation as well as real measurement data. The quantitative and qualitative results show the networks' capability, generalizability, and robustness in denoising GPR C-scans and reconstructing 3D permittivity maps of subsurface objects.

Finally, to alleviate the computational burden of the training stage of the deep learning-based inversion and to rapidly generate a large and diverse dataset, a fast deep learning-based GPR forward solver is proposed to predict the GPR B-scans of subsurface objects buried in the heterogeneous soil. The proposed solver is constructed as a bimodal encoder-decoder neural network. Two encoders followed by an adaptive feature fusion module are designed to extract informative features from the subsurface permittivity and conductivity maps. The decoder subsequently constructs the B-scans from the fused feature representations. To enhance the network's generalization capability, transfer learning is employed to fine-tune the network for new scenarios vastly different from those in the training set. Numerical results show that the proposed solver achieves a mean relative error of 1.28%. For predicting the B-scan of one subsurface object, the proposed solver requires 12 milliseconds, which is 22,500x less than the time required by a classical physics-based solver. With its excellent efficiency and generalization capability, this solver

serves as a promising forward solver for data augmentation and training.

Acronyms and Symbols

Acronyms:

GPR	Ground-penetrating radar
EM	Electromagnetic
TX	Transmitter
RX	Receiver
1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
DNN	Deep neural network
CNN	Convolutional neural network
MRF	Multi-receptive-field
MRF-UNet	U-shape network with MRF convolutions
DMRF-UNet	Double-stage U-shape network with MRF convolutions
MRF-UNet1	The first-stage MRF-UNet for denoising
MRF-UNet2	The second-stage MRF-Unet for inversion
3DInvNet	3D GPR inversion network
RE	Relative error
FWI	Full-waveform inversion
GAN	Generative adversarial network
FDTD	Finite-difference time-domain
ReLU	Rectified linear unit
SSIM	Structural similarity

MSE	Mean square error
MAE	Mean absolute error
MRE	Mean relative error
Enc-Dec	Encoder-decoder neural network
SMRF-UNet	Single-stage U-shape network with MRF convolutions
SVD	Singular value decomposition
PCA	Principal component analysis
NMF	Non-negative matrix factorization
RoI	Region of interest
MSFA	Multi-scale feature aggregation
BN	Batch normalization
MAPE	Mean absolute percentage error
CA	Cross-attention
Huber	Pseudo-Huber loss

Symbols:

X	2D subsurface scenario (permittivity distribution)
Y	GPR B-scan
$H(\cdot)$	2D GPR forward mapping
$H^{-1}(\cdot)$	2D GPR inverse mapping
Y_c	Clutter signal of GPR B-scan
Y_t	Target signal of GPR B-scan (the denoised B-scan)
η	Index of the current layer in a neural network
r_η	Receptive field size of the ηth layer
l	Index of the layers before the current layer

s_l	Stride of the convolution in the lth layer
ℓ	Total loss of DMRF-UNet
ℓ_1	Loss of MRF-UNet1 for 2D denoising
ℓ_2	Loss of MRF-UNet2 for 2D inversion
α	Weight of ℓ_1
β	Weight of ℓ_2
H_1	Height of the denoised B-scan image
W_1	Width of the denoised B-scan image
i_1	Index of H_1
j_1	Index of W_1
H_2	Height of 2D subsurface permittivity distribution
W_2	Width of 2D subsurface permittivity distribution
i_2	Index of H_2
j_2	Index of W_2
$\hat{Y}_t$	Ground-truth denoised B-scan
$\hat{X}$	Ground-truth 2D permittivity distribution
P	2D Predicted result
T	2D ground truth
H	Height of an image
W	Width of an image
i	Index of H
j	Index of W
μ_P	Mean of the 2D predicted result
μ_T	Mean of the 2D ground truth
σ_P	Variance of the 2D predicted result

σ_T	Variance of the 2D ground truth
σ_{PT}	Covariance of the 2D predicted result and ground truth
ε_r	Relative permittivity
R	Dynamic range of the image
c_1	The first variable of structural similarity
c_2	The second variable of structural similarity
$\tilde{X}$	3D subsurface scenario (permittivity map)
$\tilde{Y}$	GPR C-scan
$\tilde{H}(\cdot)$	3D GPR forward mapping
$\tilde{H}^{-1}(\cdot)$	3D GPR inverse mapping
C	Channel number in Denoiser
f_0	Initial feature map
D	Depth of a 3D image
$\mathcal{F}_{w,b}(\cdot)$	3D convolutional layer with the weight w and bias b
$\delta(\cdot)$	ReLU activation
$M_l(\cdot)$	Feature learning module
f_1	Feature map after the first residual block
f_2	Feature map after the second residual block
z	Channel-wise statistics in the feature attention block
c	Channel index
k	Index of D
$\mathcal{L}_w(\cdot)$	Fully connected layer with the weight w
$\varphi(\cdot)$	Sigmoid activation
a	Attention vector
r	Reduction ratio

f_l	Output feature map of the feature learning module
m	Index of feature learning module
$\tilde{Y}_d$	Denoised C-scan
f_{r_1}	Feature map with the first receptive field size of $r_1 \times r_1 \times r_1$
f_{r_2}	Feature map with the second receptive field size of $r_2 \times r_2 \times r_2$
f_{r_3}	Feature map with the third receptive field size of $r_3 \times r_3 \times r_3$
f_{ms}	Fused multi-scale feature map
n	Number of 3D encoding/decoding blocks
$\tilde{C}$	Number of channels in Inverter
$\hat{\tilde{Y}}_d$	Ground-truth noise-free C-scan
$\tilde{\ell}_1$	3D denoising loss
$\hat{\tilde{X}}$	Ground-truth 3D permittivity map
$\tilde{\ell}_2$	3D inversion loss
$\tilde{P}$	3D predicted result
$\tilde{T}$	3D ground truth
$\mu_{\tilde{P}}$	Mean of the 3D predicted result
$\mu_{\tilde{T}}$	Mean of the 3D ground truth
$\sigma_{\tilde{P}}$	Variance of the 3D predicted result
$\sigma_{\tilde{T}}$	Variance of the 3D ground truth
$\sigma_{\tilde{P}\tilde{T}}$	Covariance of the 3D predicted result and ground truth
x	2D subsurface scenario in GPR forward modelling

y	GPR B-scan corresponding to x
$h(\cdot)$	GPR forward mapping from x to y
x_{ε_r}	Relative permittivity map of x
σ	Conductivity
x_σ	Conductivity map of x
I	Number of encoding blocks of the forward solver
f_{E_i}	Feature map of the i^{th} encoding block
C_{E_i}	Channel number of f_{E_i}
H_{E_i}	Height of f_{E_i}
W_{E_i}	Width of f_{E_i}
F_{ε_r}	ε_r-modal feature representation
F_σ	σ-modal feature representation
Q	Query in an attention block
K	Key in an attention block
V	Value in an attention block
d	Dimension of K
Q_{ε_r}	Query of F_{ε_r}
Q_σ	Query of F_σ
A_{ε_r}	ε_r-modal attention features
A_σ	σ-modal attention features
N_h	Number of paralleled heads
$F_{\varepsilon_r \leftrightarrow \sigma}$	Fused cross-modal feature representation
J	Number of decoding blocks of the forward solver
f_{D_j}	Feature map of the j^{th} decoding block

C_{D_i}	Channel number of $f_{D\,i}$
H_{D_i}	Height of $f_{D\,i}$
W_{D_i}	Width of $f_{D\,i}$
$h_\theta(\cdot)$	Forward mapping network with the parameters θ
$\hat{y}$	B-scan generated by physics-based solver
T	Time window of forward modelling
D_s	Total depth of the subsurface scenario
L	Length of survey line (scanning distance)
δ_H	Hyperparameter of Huber loss

Chapter 1

Introduction

1.1 Motivation and Objectives

Ground-penetrating radar (GPR) has been widely used in geophysics and civil engineering due to its high-resolution, low-cost, and non-destructive characteristics [1]. Over the last few decades, GPR technology developed for subsurface detection and imaging has been popularly applied to inspect buildings [2-4], road pavements [5-6], tunnel liners [7-8], geotechnics [9-10], and underground utilities [11-13]. The working mechanism of GPR is based on the transmission and reflection of electromagnetic (EM) waves. As shown in Figure 1.1, the transmitter (TX) antenna in a GPR system emits the EM wave (signal) propagating and penetrating into the subsurface material. Variations in the electrical and magnetic properties of the background environment and the buried object cause the scattering and reflection of EM waves. The receiver (RX) antenna in the GPR system captures and detects the scattered and reflected signals as a function of time or frequency and spatial position of the scanning. The characteristics of the received signals correspond to the properties of the subsurface structure. To this end, the properties of the subsurface structure can be inversely restored from the obtained GPR data. This forms the fundamental basis for detecting visually impenetrable structures via the GPR technology.

Different TX/RX configurations can be employed for transmitting and receiving the EM signals in a GPR survey. In the monostatic configuration as shown in Figure 1.2(a), the TX and the RX are at the same location, or a single antenna is used as TX or RX moving along the scanning trajectory. In the bistatic configuration with the Common Offset (CO) setup as shown in Figure 1.2(b), the TX and RX move simultaneously along the scanning trajectory and the spatial offset between them is fixed. In the bistatic configuration with the Common Source (CS) setup as shown in Figure 1.2(c), the TX is fixed, and the RX moves along the scanning trajectory.

Similarly, the RX can be fixed, and the RX moves in the Common Receiver (CR) setup. In the bistatic configuration with the Common Midpoint (CMP) setup as shown in Figure 1.2(d), the TX and RX move in opposite directions, respectively, and the midpoint between them is fixed. In the multistatic configuration as shown in Figure 1.2(e), the TX antenna transmits signals at different positions and an array of RX antennas is used to collect signals at each position. In the cross-borehole configuration as shown in Figure 1.2(f), the TX and RX antennas are buried inside boreholes for the transmitting and receiving signals.

Figure 1.3 further illustrates the GPR forward and inverse problems with the bistatic configuration and the CO setup. In the GPR forward problems, the TX transmits EM waves into the ground. When encountered the buried object with dielectric properties different from those of the surrounding background, the EM waves are scattered and reflected, which are then recorded by the RX. The A-scan provides time domain information (t-axis) when the TX-RX pair is at a certain spatial point. The B-scan image is constructed by stacking multiple A-scans collected when TX-RX pair is moved along the scanning direction (x-axis). With multiple scanning trajectories in the xy plane, the C-scan image is obtained by stacking B-scans along y-axis. In this way, the A-scan, B-scan, and C-scan encapsulate the information of subsurface scenarios in one-dimensional (1D), two-dimensional (2D), and three-dimensional (3D) domains, respectively. However, we cannot intuitively identify the properties of the subsurface objects from the recorded GPR data. It's crucial to develop inversion algorithms to translate the GPR data into recognizable subsurface structural images. In the GPR inverse problems, these subsurface structural images, subsurface permittivity distributions, are recovered from the obtained GPR data. They provide a powerful and intuitive approach for inspecting structural integrity as these permittivity maps clearly demonstrate the properties of the subsurface objects, such as their shapes, sizes, positions, and permittivities. Those have practical significance for subsurface object identification, subterranean diagnostics, health monitoring, and subsurface utility mapping. The goal of this study is to solve the GPR inverse problems via developing novel, effective, and efficient algorithms for reconstructing subsurface permittivity maps. It should be noted that we will focus on investigating the B-scan and C-scan images because they contain more information of the subsurface scenario at the reflected signals are collected at various spatial positions.

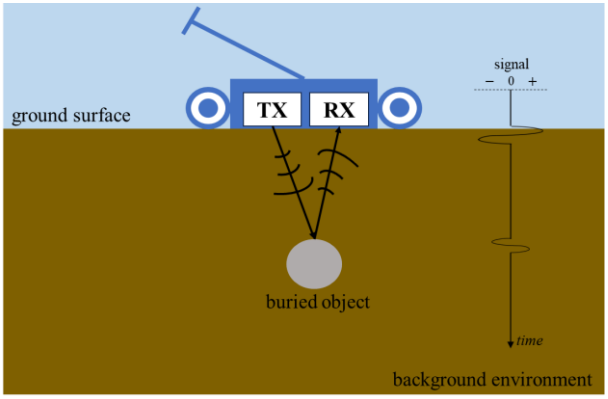


Figure 1.1: Illustration of a basic ground-penetrating radar (GPR) system. TX represents the transmitter antenna for transmitting the electromagnetic (EM) signal into the ground and RX represents the receiver antenna for recording the reflected signal from the buried object.

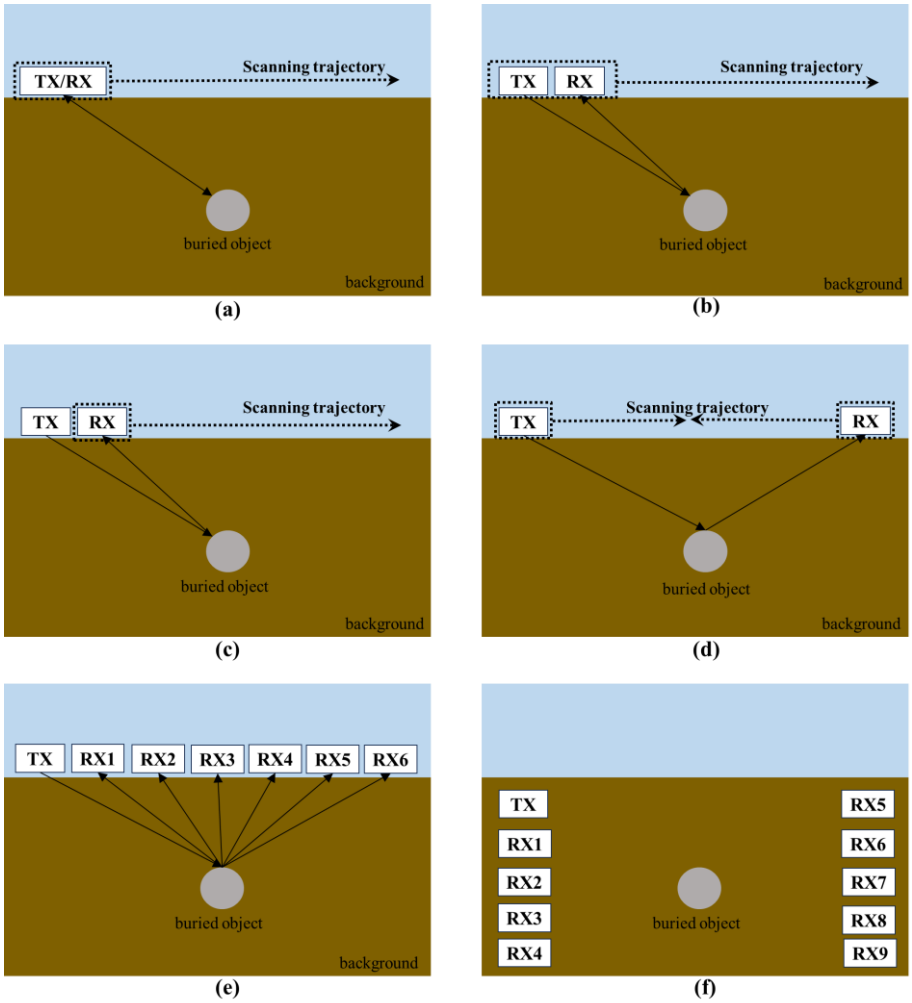


Figure 1.2: Different GPR measurement configurations. TX represents the transmitter antenna for transmitting EM waves into the ground and RX represents the receiver antenna for recording the reflected signals. (a) Monostatic configuration. (b) Bistatic configuration with Common Offset (CO) setup. (c) Bistatic configuration with Common Source (CS) setup. (d) Bistatic configuration with

Common Midpoint (CMP) with setup. (e) Multistatic configuration. (f) Cross borehole configuration.

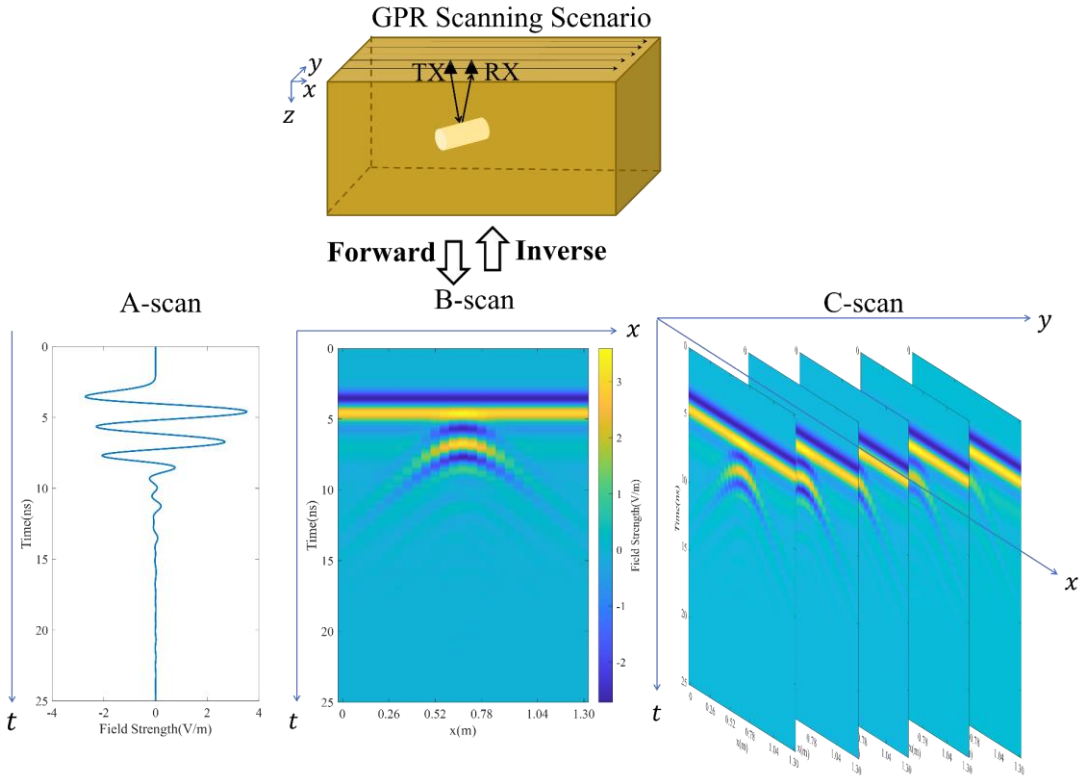


Figure 1.3: Illustration of GPR forward and inverse problems. TX represents the transmitter for transmitting the EM signal and RX represents the receiver for recording the reflected signal from the subsurface scenario.

In recent years, many inversion techniques have been developed to reconstruct subsurface structural images from GPR data [10, 14-15]. However, these traditional iterative algorithms suffer from high computational costs and low accuracy when applied to complex subsurface scenarios. To improve the inversion efficiency and accuracy, deep learning-based methods have been proposed for the GPR applications, such as hyperbolic signature recognition, image classification, and object detection [16-17]. To restore the subsurface permittivity distributions, DNNs have been utilized to find the mapping relationships between the GPR data and permittivity maps [18]. During the training phase of these networks [Figure 1.4(a)], a set of GPR data and corresponding subsurface permittivity maps are used to train the DNN. A loss function is employed to optimize the network and find the minimal misfit, then the parameters of the DNN will be updated. In the testing phase of these networks [Figure 1.4(b)], using the well-trained DNN to describe the optimal mapping relationship

between GPR data and permittivity maps, the permittivity maps can be predicted from new GPR data automatically and rapidly.

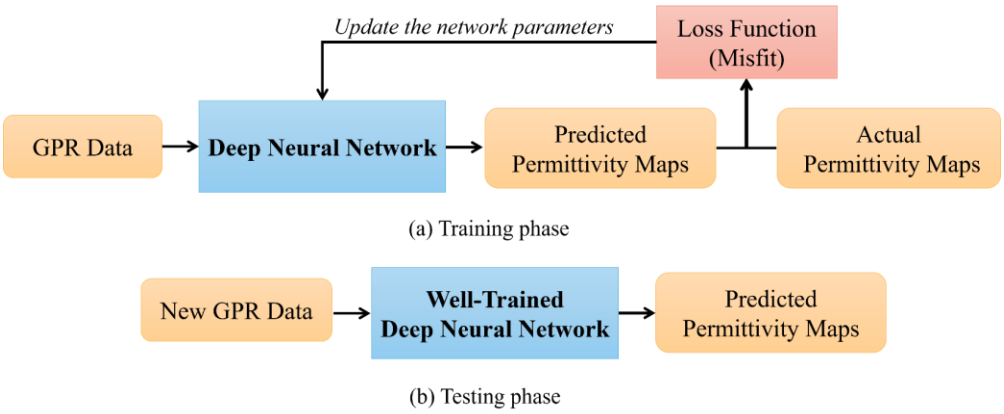
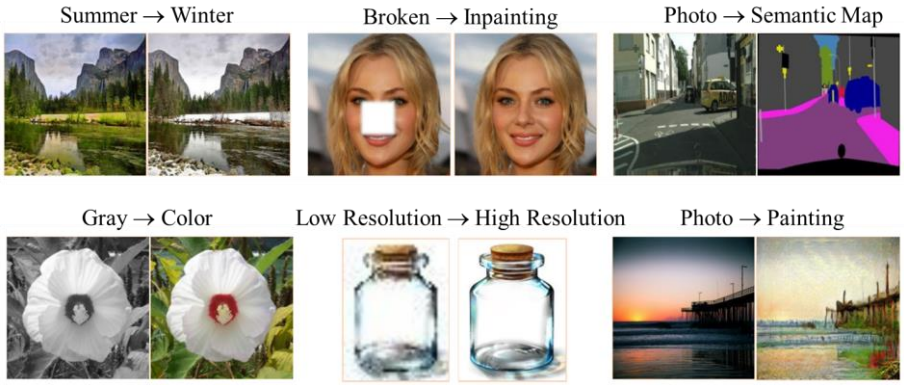
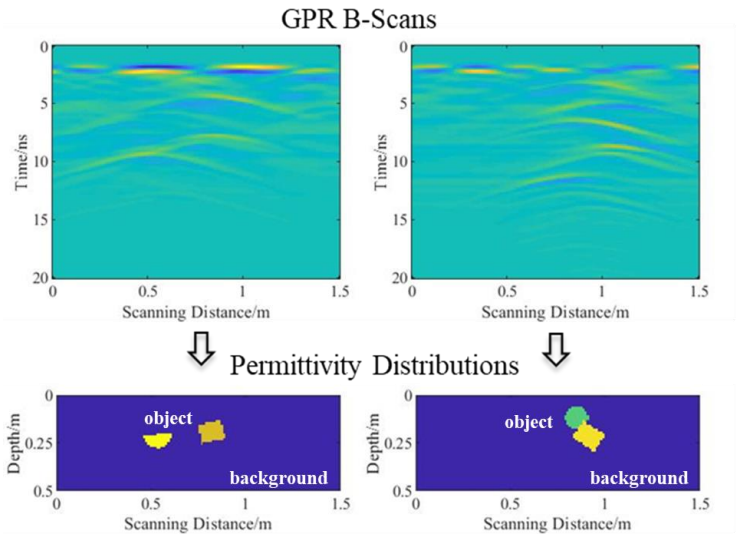


Figure 1.4: Basic framework of permittivity map reconstruction via deep learning techniques, including the (a) training phase and (b) testing phase.

The main challenges of the permittivity map reconstruction based on deep learning are as follows: 1) As shown in Figure 1.5(a), the majority of existing DNN-based image-to-image translation models [19] are designed for spatially aligned data pairs, while there is no specific spatial alignment and pixel-to-pixel correspondence between the GPR scans and permittivity maps as shown in Figure 1.5(b). A DNN model with high non-linearity is required for specifically describing an accurate mapping relationship from GPR scans to permittivity maps. 2) Complex subsurface scenarios often yield complicated reflection patterns in the obtained scans. As shown in Figure 1.5(b), when several objects with different properties are close to each other, their hyperbolic reflection patterns are overlapped or obscured, which makes it hard to differentiate the objects' signatures and perform the subsequent permittivity distribution reconstruction. 3) The heterogeneity of the subsurface environment introduces clutter and noise in the scans that interfere with the signatures of reflections from objects. Especially when the object's permittivity is close to that of the surrounding soil, the reflected field strength from the object is weak. Thereby, the reflection from the object can be easily disguised by environmental clutter and noise. In this study, we aim to solve these challenges and apply deep learning techniques for effectively and efficiently reconstructing subsurface permittivity maps from the obtained GPR data.



(a)



(b)

Figure 1.5: (a) Examples of classical DNN-based image-to-image translation [19]. (b) Examples in our task that aims to transform the GPR images to the corresponding subsurface permittivity distributions.

1.2 Major Contribution of the Thesis

In this thesis, we propose two deep learning-based inversion approaches for reconstructing subsurface permittivity maps from GPR data and one forward solver for alleviating the computational burden of generating large datasets for training deep learning-based inversion models. The major contributions are summarized as follows.

(1) First, a two-stage DNN, named DMRF-UNet, is proposed to address the issues in the permittivity distribution reconstruction of subsurface objects under heterogeneous

soil conditions. The DMRF-UNet consists of double U-shape convolutional neural networks (CNNs) with multiple receptive fields (MRF-UNet) to describe the representation of the GPR inverse problem. The first MRF-UNet (MRF-UNet1) extracts the hyperbolic signatures of reflections from subsurface objects and removes the clutter and noise due to the heterogeneous soil in the input noisy B-scan. The denoised B-scan and the noisy B-scan are concatenated as input to the second MRF-UNet (MRF-UNet2). The MRF-UNet2 learns the mapping relationship between the signatures in the two-channel B-scans and the permittivity distribution of subsurface objects. In two stages, convolutions with multiple receptive fields are utilized to extract multi-scale hyperbolic signatures corresponding to the reflections from subsurface objects in the B-scans. The end-to-end training is conducted based on the combined loss from two stages to avoid information loss. A dataset including the noisy B-scans obtained under heterogeneous soil conditions, the denoised B-scans, and the permittivity distributions of subsurface objects are generated to train and test the DMRF-UNet. The comparative results on numerical and real measurement data demonstrate the superior performance of the proposed method over existing studies. The main novelties of this work are as follows:

- This is the first time that a deep learning-based framework is proposed for reconstructing the permittivity distributions of subsurface objects under heterogeneous soil conditions.
- A unique two-stage DNN which outperforms existing works for the reconstruction under heterogeneous soil conditions is carefully designed.
  - A novel prior denoising network is introduced to extract the objects' signatures and eliminate noise interference under heterogeneous soil conditions.
  - The DNN employs multiple receptive fields to extract multi-scale hyperbolic signatures corresponding to the reflections from multiple objects with various properties in the B-scan. Cascaded small-size convolutional filters are specifically used instead of large-size filters to reduce the computation cost.
  - The two-stage DNN is trained end-to-end with a combined loss function to balance the training of the two stages and avoid information loss. The

proposed network that combines all these components achieves the best performance for reconstructing permittivity distributions under heterogeneous soil conditions.

(2) Second, we propose a deep learning scheme, called 3DInvNet, that reconstructs subsurface 3D permittivity maps from GPR C-scans with prior denoising. To the best of our knowledge, this is the first work that performs the 3D subsurface scenario reconstruction from GPR data via a deep learning scheme. The main contributions of the proposed scheme compared to the existing deep learning based GPR detection/2D reconstruction schemes are summarized as follows.

- A prior 3D denoising network, called Denoiser, is designed to suppress noise in GPR C-scans due to heterogeneous soil environments. Different from complicated networks in existing GPR data denoising networks, the Denoiser adopts a small 3D CNN with residual learning and feature attention mechanism to extract subsurface objects' reflection signatures in noisy C-scans effectively.
- After denoising, a 3D U-shaped encoder-decoder network, called Inverter, is designed to map the denoised C-scans predicted by the Denoiser into subsurface 3D permittivity maps. Instead of using conventional convolutions with a single scale, multi-scale feature aggregation modules are proposed to extract the features of multiple objects with various properties.
- A three-step separate learning strategy is used to pre-train and fine-tune the Denoiser and Inverter.

Test results show that the proposed scheme is capable of denoising C-scans and then reconstructing 3D permittivity maps with very high accuracy and efficiency. The comparative results demonstrate the improved accuracy of reconstructing 3D permittivity maps from the denoised C-scans rather than the noisy C-scans. The test results on new scenarios (not included in the training/testing dataset) show good generalizability and robustness of the proposed scheme.

(3) Third, a fast GPR forward solver for obtaining 2D radar signatures of subsurface convex objects is proposed to alleviate the high computational cost of large dataset generation for training deep learning-based inversion networks. A deep encoder-decoder neural network is designed to predict the GPR B-scan for given permittivity

and conductivity maps of the subsurface scenario. The bimodal encoder of the proposed network independently extracts informative features from these two input maps. To enhance the cross-modal connectivity, a feature fusion module with cross attention is employed to adaptively fuse the extracted bimodal feature representations. The fused feature representations are then fed into the decoder to produce the B-scan. The network is trained with a diverse set of subsurface permittivity and conductivity maps and their corresponding B-scans under heterogeneous background soil conditions. Transfer learning is introduced to boost the network's generalization capability for new scenarios vastly different from those in the training set. The test results show that the proposed solver achieves an average relative error (RE) of 1.28% and a near-real-time B-scan prediction in 12 milliseconds.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows.

Chapter 2 provides a literature review of related works. First, traditional 2D/3D GPR inversion algorithms are introduced. Then, recent studies on deep-learning-based GPR inversion are expounded. The limitations of these methods are indicated. Finally, existing GPR forward modeling methods are presented.

Chapter 3 introduces the proposed scheme for 2D GPR data inversion based on deep learning. The detailed methodology, including the two-stage 2D network structure and the loss function for training the network end-to-end, is described. Experiments on simulation and real measurement data are presented. The inversion results are qualitatively and quantitatively analyzed and a comparative study with existing methods is provided.

Chapter 4 describes the proposed approach for 3D GPR data inversion based on deep learning. In the methodology, the structures of the first-stage 3D denoising network and the second-stage 3D inversion network are introduced; the proposed three-stage learning strategy is explained in detail. The results of the simulation and real measurement data are analyzed and discussed.

Chapter 5 presents the proposed fast GPR forward solver for alleviating the computational burden of large dataset generation in deep learning-based inversion algorithms. The bimodal network structure that considers both the permittivity and

conductivity is described and the transfer learning for improving the network's generalizability is introduced. The accuracy and efficiency of the proposed method are compared with the traditional physics-based solver.

Chapter 6 gives the conclusions and recommendations for future works.

Chapter 2

Literature Review

In this chapter, the literature reviews on traditional 2D/3D GPR inversion algorithms, deep-learning-based GPR inversion approaches, and existing GPR forward solvers are provided. The advantages and disadvantages of these studies are discussed.

2.1 Traditional GPR Inversion

2.1.1 GPR Data Pre-Processing

After obtaining the GPR data as shown in Figure 1.3, pre-processing algorithms are often used to initially improve the quality of GPR data and increase the visibility of object signatures. The basic GPR data pre-processing algorithms, including (1) time-zero correction, (2) time-varying gain, (3) mean subtraction, and (4) singular value decomposition (SVD), are introduced as follows.

First, time-zero correction has been developed for conducting accurate depth measurements [20-21]. In the GPR scanning process, several factors such as the different air temperature, connecting cable length, and antenna height at different scanning point will result in the inconsistent position of the reflection coming from the subsurface object in the received A-scan signal. To avoid this issue, we can select a common zero time point by cutting the A-scan signals to a fixed threshold. The position of the threshold can be the first break point, the first negative peak, the first negative peak, or the zero-amplitude point between the negative and the positive peaks of each A-scan. The positions are demonstrated in Figure 2.1, which also shows one example of selecting the first break point as cutting threshold for each A-scan and the resulting B-scan.

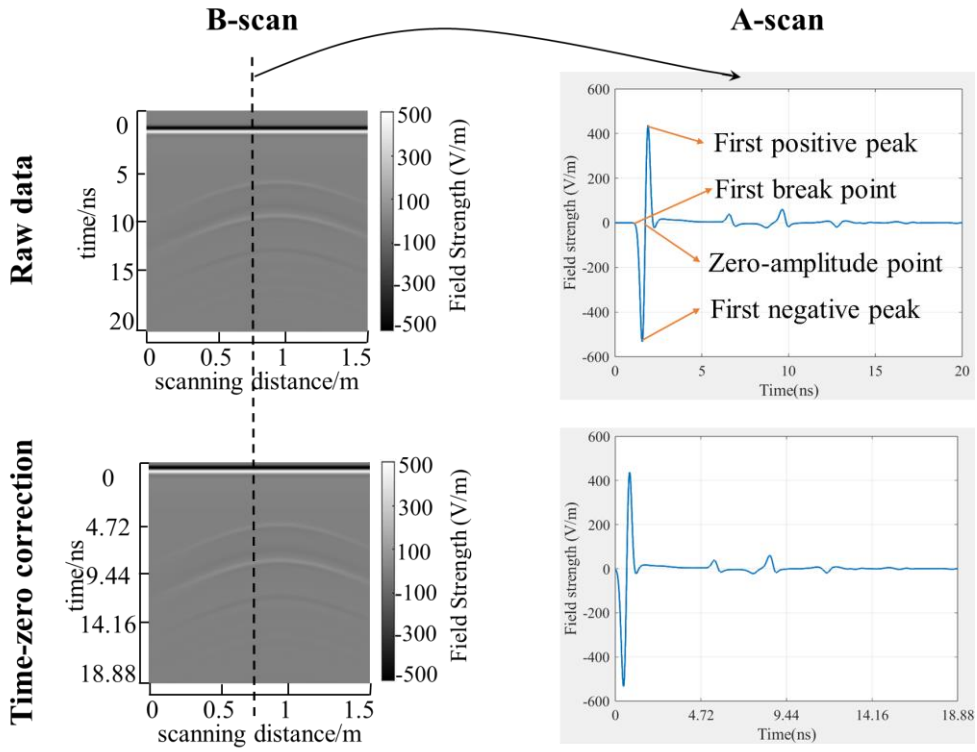


Figure 2.1: One example of time-zero correction for GPR A-scans and B-scans in which the first break position is set as the time-zero.

Second, time-varying gain has been developed to compensate the signal attenuation due to the EM wave propagation in a lossy medium and the geometrical spreading of the energy [1, 21]. The time-varying gain is applied to each A-scan to improve the visibility of the object signatures. The expression of the time-varying gain is:

$$E_g(t) = E(t) \cdot k(t) \cdot t, \tag{2.1}$$

where t is the time, $E(t)$ is the raw A-scan, $k(t)$ is the gaining function which can be linear or exponential, and $E_g(t)$ is the processed A-scan after the time-zero correction. However, the performance of time-varying gain depends on the selection of the gaining function and the great change of the field strength as shown in Equation (2.1) will affect the accurate restoration of the subsurface object's properties.

Third, mean subtraction is used to remove the direct coupling and the ground reflection by leveraging their common characteristics [21-22]. The expression of the mean subtraction algorithm can be two-step:

$$\dot{E}(x,t) = E(x,t) - \frac{1}{N_t}\sum_{i=1}^{N_t} E(x,i)\,, \tag{2.2}$$

$$E_m(x,t) = \dot{E}(x,t) - \frac{1}{N_x}\sum_{i=1}^{N_x} \dot{E}(i,t)\,, \tag{2.3}$$

where t is the time, x is the distance along the scanning trajectory, $E(x,t)$ is the received raw B-scan, N_t is the number of time steps, N_x is the number of scanning traces, and $E_m(x,t)$ is the processed B-scan after the mean subtraction. The first step in Equation (2.2) is to make the mean of the received B-scan over time is zero, and the second step in Equation (2.3) further subtracts the mean over the scanning traces. Figure 2.2 shows one example of using mean subtraction to pre-process the raw B-scan. We can observe that the direct coupling and the reflections from the ground are effectively removed and the reflections from the subsurface object are retained.

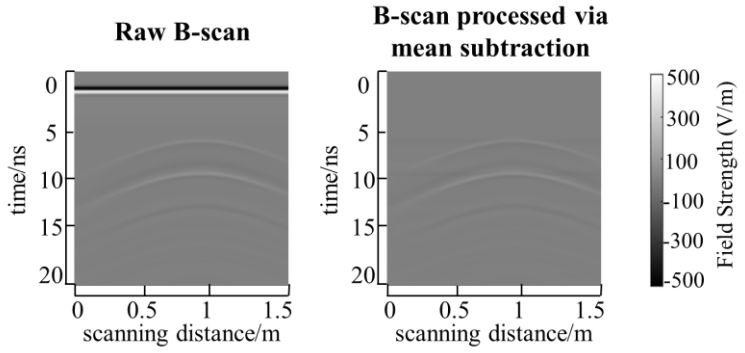


Figure 2.2: One example of using the mean subtraction to remove the direct coupling and ground reflections in a B-scan.

Fourth, SVD is a subspace projection method introduced to pre-process the GPR B-scan for noise removal [22-23]. Assuming a B-scan is a 2D matrix E with the dimension of $N_t \times N_x$, it can be expressed as

$$E(x,t) = USV^T, \tag{2.4}$$

where U with the dimension of $N_t \times N_t$ and V with the dimension of $N_x \times N_x$ are orthogonal matrices containing eigenvectors. S with the dimension of $N_t \times N_x$ is a diagonal matrix containing the eigenvalues $\lambda_{i\in[1,\min(N_t,N_x)]}$. The processing using SVD remains the eigenvalues related to object reflections while the rest related to

noise are set to zero. Assuming the noise such as the direct coupling and the ground reflections in a B-scan is much stronger than the object reflections, the noise subspace will be related to the highest eigenvalue of the B-scan matrix. The processed B-scan $E_s(x,t)$ removing the first $\grave{N}_x$ eigenvalues can be expressed as

$$E_s(x,t) = U \cdot S\big(1{:}\,N_t, (\grave{N}_x + 1){:}\,N_x\big) \cdot V\big(1{:}\,N_x, (\grave{N}_x + 1){:}\,N_x\big)^T. \tag{2.5}$$

Figure 2.3 presents one example of processing a B-scan via SVD using different values of $\grave{N}_x$. We can observe that the result with $\grave{N}_x = 1$ is the best and similar to the result of mean subtraction [Figure 2.2], and the object reflections are even corrupted as $\grave{N}_x$ increases. Thus, one weakness of SVD is the high dependence on the selection of eigenvalues.

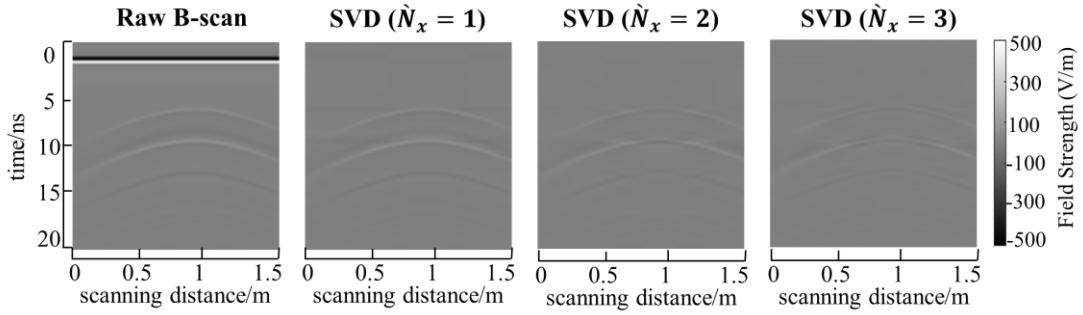


Figure 2.3: One example of processing the raw B-scan via singular value decomposition (SVD) using different values of $\grave{N}_x$, where $\grave{N}_x$ is the number of the removed eigenvalues.

2.1.2 2D GPR data inversion

To solve 2D GPR inverse problems, there exists several classical methods for reconstructing subsurface structural images from the GPR B-scans, including migration algorithms, tomographic approaches, and full-waveform inversion (FWI) techniques.

Migration algorithms. In the past years, migration algorithms have been developed for transforming the unfocused space-time GPR image to a focused one, showing the object's location and size with corresponding EM reflectivity [24]. Assuming a point-like scatterer in the $x-d$ object domain is located at (x_s, d_s), the scattered field is recorded at $(x, 0)$, and the transmitted signal is $s_T(t)$, the resulting diffraction hyperbola emerges in the obtained B-scan image, which is in the $x-t$ data domain.

When the wave propagation speed in the soil background is v, the apex of the hyperbola should be located at $(x_s, 2d_s/v)$ in the B-scan of the data space. Migration algorithms aim to collapse such hyperbola at its apex in the object domain as shown in Figure 2.4. At each scanning point, the scatterer should reside on a semicircle of radius equal to half the product of the travel-time and the wave-speed in the soil, resulting in that each data point in the data domain is equally distributed over a semicircle in the object space so that all the semicircles intersect at (x_s, d_s) [25].

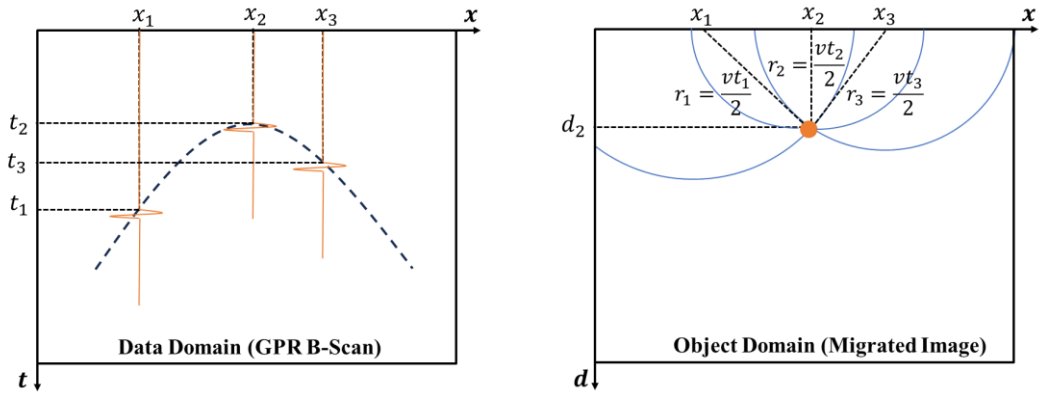


Figure 2.4. Illustration of GPR data migration process. $\boldsymbol{x}$, $\boldsymbol{t}$, and $\boldsymbol{d}$ represent the distance along the scanning trajectory, time, and depth of the subsurface object.

To achieve such migration process, several algorithms have been proposed. First, diffraction summary algorithm divided the object domain in pixels and a diffraction hyperbola is constructed in the data domain for each of the pixels. Inversely, the reconstruction at each pixel was obtained by summing up all the A-scans that the hyperbola intersects [25]. Second, the reverse-time migration was proposed based on the two-way wave equation [26]. This algorithm usually comprises three essential steps: (1) the forward propagation of the EM field at the source point, (2) the reverse extrapolation of the EM wave at the receiving point, and (3) utilizing the relevant imaging conditions to obtain migrated profiles based on the forward and backward wave fields at the same time [27]. Third, Kirchhoff migration built the migrated profile via finding the solution to the scalar wave equation for the wave function in the subsurface medium [28]. Forth, frequency-wavenumber migration started from the wave equation involving the exploding source model and determined the field in the object domain by back-propagating the field measurements [25, 29]. However, all these migration algorithms did not provide the constitutive parameters of the object,

of great importance for identifying subsurface objects and monitoring their conditions.

Microwave tomography. Microwave tomography approaches were investigated to solve EM inverse scattering problems and reconstruct the subsurface 2D dielectric profiles from the acquired GPR data. In [30], a microwave tomographic approach relying on a distorted wave scattering model that considers the presence of the buried pipe within the background scenario was exploited to monitor an early-stage water leak from a metallic buried pipe using GPR data. In [31], a microwave inverse scattering algorithm based on the Born approximation (BA) and exploiting a differential measurement configuration for obtaining the GPR data was proposed to reconstruct the subsurface object. In [32], an innovative GPR tomography technique integrating a customized multifrequency version of the particle swarm optimizer (PSO) within the iterative multi-scaling approach (IMSA) was proposed to solve the nonlinear inverse scattering problem and retrieve dielectric profiles of the subsurface target with various shapes and types. In [33], a linearized microwave tomographic approach with GPR measurements based on the BA-based formulation of the inverse scattering problem was investigated for the detection, localization, and rough estimation of the shapes of the targets in order to monitor the cultural-heritage and archaeological sites. However, the requirements of the measurement configuration in the microwave tomography methods are often high, and the iterative inverse scattering process is computationally expensive.

FWI algorithms. To further improve the accuracy and efficiency of 2D GPR data inversion and subsurface imaging, FWI algorithms were proposed to reconstruct subsurface 2D permittivity maps by mapping the subsurface structure information from GPR data via a non-linear least-square optimization process [14-15, 34-36]. The basic framework of the FWI algorithm is shown in Figure 2.5. It can be observed that the design of the GPR forward solver and the optimization algorithm as well as the selection of the initial model are key factors of the FWI algorithm. In [14], to detect the defects in underground concrete structures, a probabilistic FWI method was explored to determine the relative permittivity distribution of the underground structures from cross-hole GPR waveform data. In [15], a GPR FWI method, which involves the finite element time-domain method to perform GPR forward modelling and the total-variation model constraint with multi-scale inversion strategy to execute the conjugate gradient algorithm for optimization, was proposed to reconstruct the

permittivity and conductivity maps of tunnel lining defects. In [34], to tackle the non-linearity problem of FWI, a gradual and progressive expansion of the frequency content as the iterations proceed was integrated to the full-waveform time-domain inversion scheme for GPR data. In [35], to further improve the imaging resolution and accuracy of FWI, a GPR multiscale dual-parameter inversion scheme based on total variation regularization in the time domain was proposed to map the subsurface permittivity and conductivity distributions. In [36], a quasi-Newton optimization scheme was employed in the FWI algorithm with a Tikhonov regularization for the simultaneous reconstruction of permittivity and conductivity in 2D scenarios. However, all these FWI algorithms need to perform the forward modelling in every iteration, while the optimization usually requires a large number of iterations. This results in high computational cost. Besides, the reconstruction accuracy is often low when these algorithms are used to reconstruct the permittivity maps of complex subsurface scenarios.

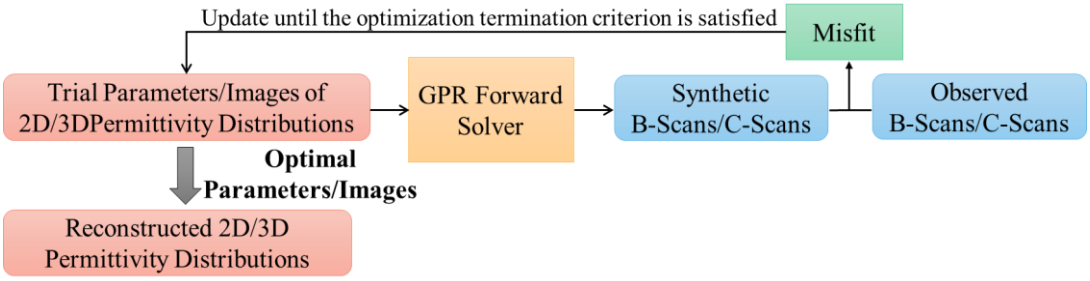


Figure 2.5: Basic framework of full-waveform inversion (FWI) algorithm.

In short, the limitations of the above conventional 2D GPR data inversion methods are listed as follows: (i) They are not capable of restoring the objects' constitutive parameters, or (ii) they require high computational cost when iteratively constructing the permittivity maps, and (iii) they provide inaccurate results when applied to permittivity map reconstruction of complex subsurface scenarios.

2.1.3 3D GPR data inversion

To further address 3D GPR inverse problems, several traditional algorithms were proposed to reconstruct 3D subsurface images from GPR C-scans, such as the back-projection algorithm [37-39], Kirchhoff migration [40], and reverse time migration [41-42]. However, similarly to 2D migration, these 3D migration algorithms can only

provide an approximation to the objects' positions and shapes but cannot reconstruct their permittivity maps, while the permittivity information is of importance for object identification and health examination. To reconstruct the subsurface permittivity map, 2D FWI algorithms were proposed in [14-15, 34-36]. However, due to its high computational cost for processing 3D GPR data, most of the current FWI studies only focus on 2D cases. To date, there are two studies that apply FWI to reconstruct 3D subsurface permittivity or conductivity distribution from GPR data based on the iterative framework shown in Figure 2.2. In [43], an FWI algorithm with total variation regularization and Hessian approximation was adopted. It took about a week to generate one 3D permittivity map of a small 3D domain. To improve the efficiency of 3D FWI, a modified total variation regularization scheme was proposed in [44]. However, the inversion process still took about 16 hours to produce a 3D permittivity map. Besides, only subsurface objects with simple and regular geometries were considered while applying 3D FWI algorithms. The high computational complexity and restricted generalizability of the 3D FWI algorithms greatly limit their applications in reconstructing complex 3D subsurface scenarios.

2.1.4 Limitation Summary

Overall, the limitations of traditional 2D and 3D GPR data inversion algorithms are summarized as follows:

1) Lack of the restoration of the objects' constitutive parameters.
2) High computational costs due to iterative optimizations.
3) Low accuracy when applied to complex subsurface scenarios.
4) Limited applications for reconstructing 3D scenarios.

To address these issues, it is crucial to further explore efficient and effective inversion schemes for reconstructing subsurface permittivity maps from the GPR data.

2.2 Deep Learning-based GPR Inversion

In recent years, deep learning techniques have been introduced to solve GPR inverse problems [16-17], including GPR image classification, object detection, restoration of the subsurface objects' properties, and subsurface permittivity map reconstruction.

These techniques have demonstrated their capability to learn the non-linear mapping between input data and the desired outputs for GPR application. The details of existing deep learning based GPR inversion techniques are described as follows.

2.2.1 GPR Image Classification

As shown in Figure 2.6, using the obtained GPR images such as the B-scan or the C-scan as inputs, DNNs were developed to classify them into different classes for restoring some information of the subsurface scenario. In [45], a 2D CNN was designed to extract meaningful signatures from GPR B-scans and classify them into buried explosive hazards and harmless false alarms. The network structure consists of two convolutional layers followed by down-pooling layers for extracting features and one fully connected layer with the non-linear activation function followed by a two-way classification layer to estimate the class of the input B-scan. To provide clear illustrations for these processes, Figure 2.7 shows the convolution operation in the convolutional layer. The down-pooling operations including the max-pooling and average-pooling are illustrated in Figure 2.8. One example of the fully connected layer is shown in Figure 2.9. Classical activation functions including the linear activation (Linear), Rectified Linear Unit activation (ReLU), Leaky Rectified Linear Unit activation (Leaky ReLU), Exponential Linear Unit activation (ELU), Tanh activation (Tanh), and Sigmoid activation (Sigmoid) are presented in Figure 2.10. In [46], a deep learning scheme was proposed to measure the moisture content status of subsurface soil. Two CNNs including a classification network and a regression network were designed for moisture quantitative classification and continuous variable prediction of water content, respectively. The high accuracy of estimating water content provided potentials for the applicability of roadbed maintenance. In [47], a CNN with a support vector machine (SVM) classifier was designed for the classification of buried objects, shape type and soil type from the input GPR B-scans. The numerical experiments and real measurements demonstrated the effectiveness and efficiency of the proposed method for recognizing the subsurface object and soil type. In [48], the combined information in B-scans and C-scans were utilized for the classification of the underground object type in urban area including the cavity, pipe, manhole, and subsoil background using a five-layer CNN. These studies verified the applicability of deep learning techniques especially the CNNs for GPR image

classification, however, only partial information of the subsurface scenarios could be restored using the classification networks and only single-object scenarios were considered.

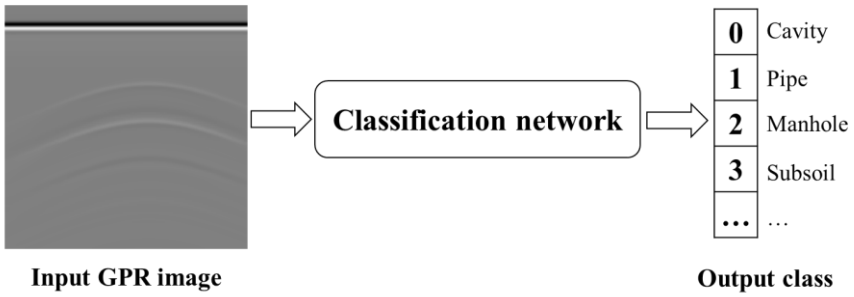


Figure 2.6: Illustration of classifying GPR images using a classification network.

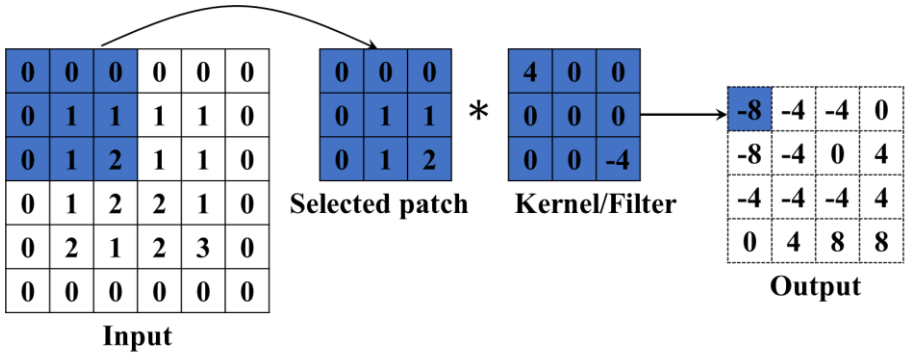


Figure 2.7: Illustration of convolution operation in the convolutional layer of convolutional neural networks (CNNs).

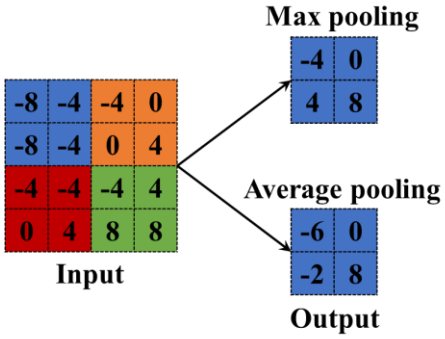


Figure 2.8: Illustration of the down-pooling layer of CNNs.

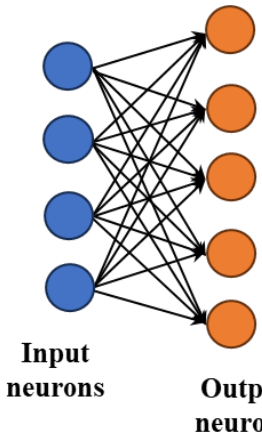


Figure 2.9: Illustration of one fully connected layer.

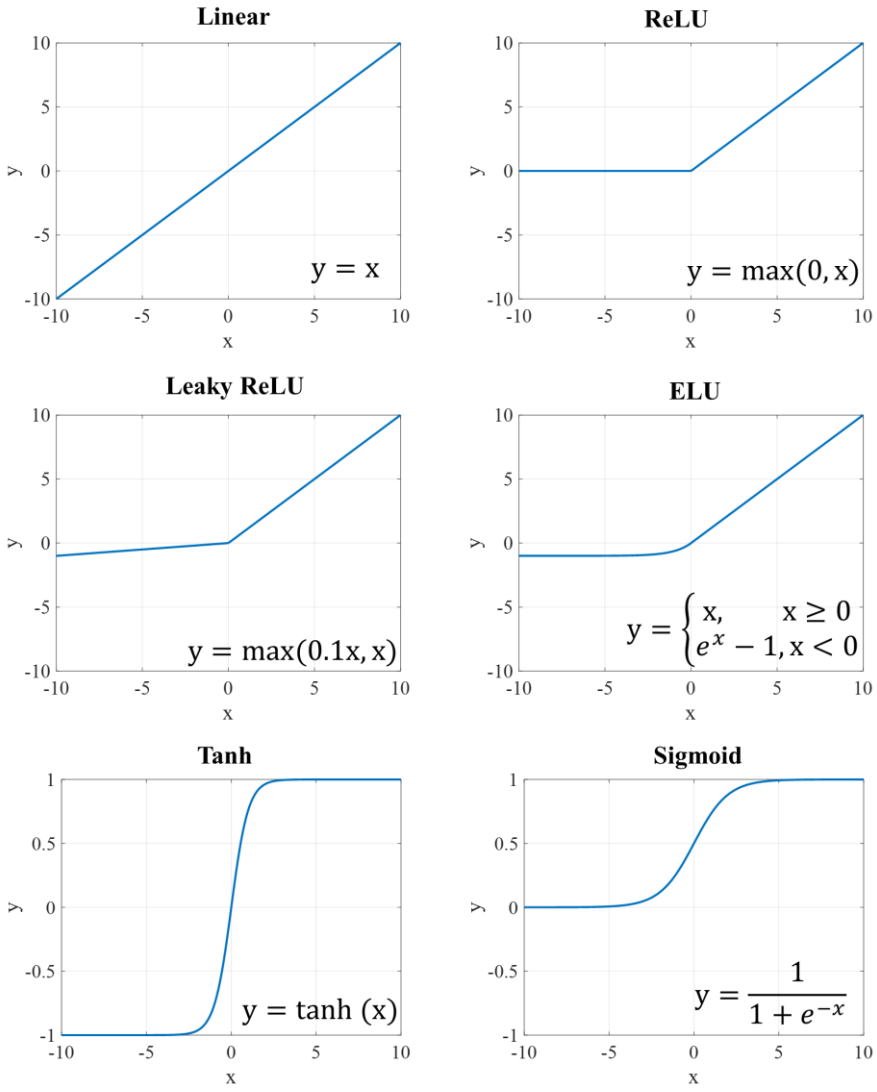


Figure 2.10: Classical activation functions. “Linear”, “ReLU”, “Leaky ReLU”, “ELU”, “Tanh”, and “Sigmoid” represent the linear activation, Rectified Linear Unit activation, Leaky Rectified Linear Unit activation, Exponential Linear Unit activation, Tanh activation, and Sigmoid activation, respectively.

2.2.2 Object Detection

To further recognize the subsurface object even for multiple-object scenarios from GPR images, object detection networks were introduced in [49-51] to predict the region and probability of every object as shown in Figure 2.11. In [49], the Faster-RCNN [52] framework was employed to detect the hyperbolic patterns reflected from the subsurface objects in a B-scan. The CNN was first pre-trained on the grayscale Cifar-10 database and then reused in the Faster-RCNN to detect hyperbolas in real and simulated GPR B-scans. In [50], the Mask Scoring R-CNN [53] architecture was developed to detect and segment hyperbolic signatures from GPR B-scans. Real-field measurements validated the effectiveness of detecting underground tree roots from the collected B-scans. In [51], the YOLOV3 [54] model was used to detect internal cracks in asphalt pavement from GPR B-scans with enhanced crack features. However, training these detection networks required a large set of B-scans with manually labelled regions such as the green box as shown in Figure 2.11. More importantly, although the detection networks can predict the types of multiple objects, the restored information for the subsurface objects is still limited.

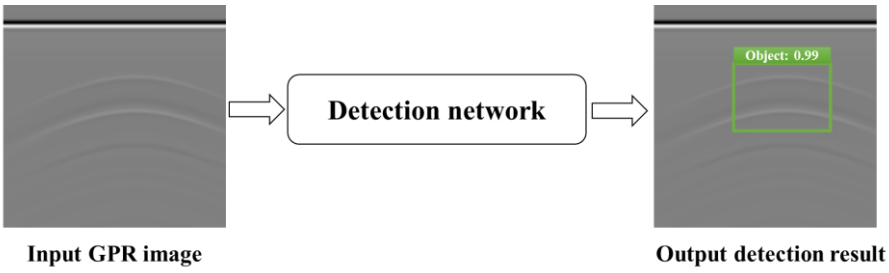


Figure 2.11: Illustration of detecting subsurface objects from GPR images using a detection network.

2.2.3 Subsurface Property Estimation

To obtain more information of the subsurface object, deep learning techniques were developed to estimate the subsurface objects' properties [55-56], in which regression networks were designed to predict continuous values of the properties as shown in Figure 2.12. In [55], a multi-polarization aggregation and selection neural network was proposed to take the multi-polarimetric GPR B-scans as inputs, integrate their characteristics in the feature space, and select discriminative features of reflected

patterns for estimating the horizontal orientation angle and vertical inclination angle of an elongated subsurface object. In [56], a 2D CNN was designed to extract the feature hyperbola of the underground target from the input B-scan and then the buried depth of the target and the relative permittivity of the background medium were calculated from the extracted features. Using the target scattered field intensity, the background permittivity, and the target burial depth as inputs, a 1D DNN was employed to estimate the relative permittivity of the subsurface target. However, only partial properties of the subsurface object were restored in these studies and their applicability for complicated multiple-object scenarios was limited.

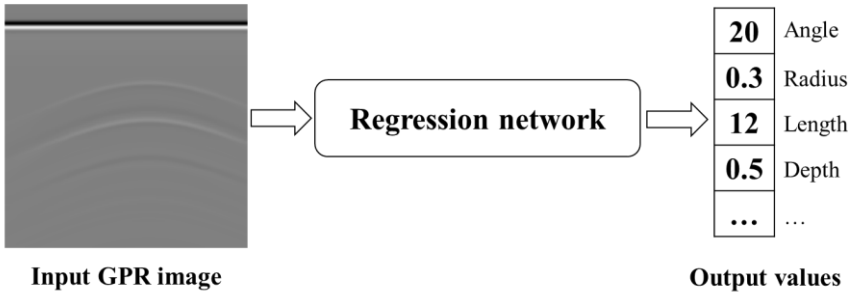


Figure 2.12: Illustration of estimating subsurface objects' properties from GPR images using a regression network.

2.2.4 Permittivity Map Reconstruction

As shown in Figure 2.13, for the restoration of subsurface permittivity distributions including the shapes, sizes, positions, and permittivity values of subsurface objects, DNNs developed for image-to-image translation have been investigated to reconstruct 2D subsurface permittivity maps from GPR B-scans [18, 57-59]. In [18], three types of DNNs for image-to-image transformation, including the encoder-decoder [60], U-Net [61], and generative adversarial network (GAN) [62], were employed to reconstruct the subsurface permittivity maps of sewer crowns. The testing results showed that the encoder-decoder and U-Net outperformed GAN. This study proved the possibility of transforming the non-intuitive B-scans into easy-to-interpret subsurface images. In [57], the U-Net with instance normalization was adopted to map the permittivity distribution from recorded GPR data in real time. In [58], a trace-to-trace encode-decoder network GPRInvNet was proposed to deal with complex GPR B-scans of tunnel linings. Based on that, to address attenuation-

induced issues and extract sufficient global information, an improved network PINet was proposed in [59] to reconstruct the permittivity map of geo-structures from B-scans. The comparative results on both numerical simulation and real measurement data in [58] and [59] demonstrated the superior performance of the deep learning-based methods compared to conventional methods. The PINet achieves state-of-the-art performance in accurately reconstructing permittivity distributions. However, these studies are limited in two aspects:

- All these deep learning-based methods only considered the inversion for homogeneous environments and the obtained B-scans contain little environmental clutter and noise. Only the study in [58] randomly cropped the noise patches obtained from the real heterogeneous background and added them to synthetic B-scans for network training. Yet, the reconstruction accuracy is greatly reduced, and the network can only reconstruct rough profiles of the subsurface objects. This is because the noise and clutter from the heterogeneous background interfere with the signatures corresponding to reflections from the objects and make the identification of reflection patterns in B-scans difficult, rendering the GPR inversion task more challenging in complex heterogeneous background conditions. Especially, when the object's permittivity is close to the surrounding soil or several objects are in close proximity to each other, the hyperbolic signatures of the objects will be much harder to recognize and even masked by the environmental clutter.

- All the works mentioned above only focus on the 2D-domain permittivity map restoration, which can only contribute to the sectional information of the subsurface scenarios. Information such as the orientation and shape of the object cannot be retrieved especially when the subsurface object is complex, and one cross-section is not sufficient to generate the full profile of the object. Furthermore, the full-wave simulations used for 2D inversion cannot capture the actual EM phenomena in 3D real world. In 2D modelling, while the scattering is assumed to be invariant in one coordinate direction, the line sources used as excitations are assumed to be infinitely long and the polarization can only be linear. However, these are not the cases in realistic 3D modelling and the results obtained by 2D modelling could be totally different from the actual response of 3D modelling [43]. Reconstructing 3D

permittivity maps via 3D full-wave simulations can address the limitations of 2D reconstruction. However, it cannot be directly achieved via the aforementioned works. Therefore, it is imperative to investigate deep learning methods for reconstructing 3D subsurface permittivity maps.

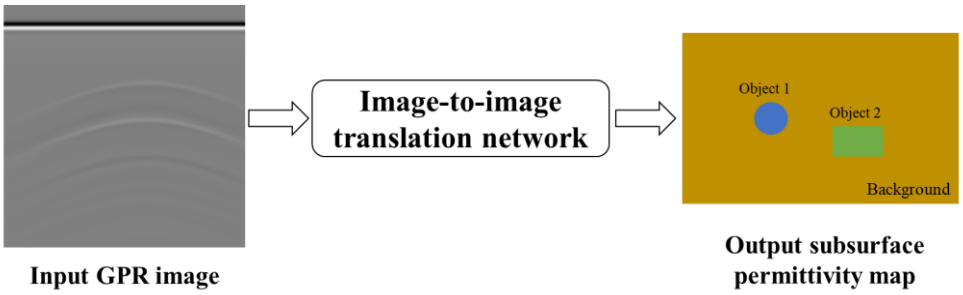


Figure 2.13: Illustration of predicting subsurface permittivity maps from GPR images using image-to-image translation networks. In the output permittivity map, the value of each pixel represents the permittivity value.

2.3 GPR Forward Solver

2.3.1 Existing Simulation Software

The deep learning-based inversion networks for reconstructing the subsurface permittivity maps from GPR images require large sets of numerical simulation data for training and testing. An effective and efficient GPR forward solver should be used to generate the datasets. For this purpose, we investigate existing GPR simulation software including GPRSIM [63], Computer Simulation Technology (CST) [64], and gprMax [65-66]. The details are introduced as follows.

GPRSIM has been developed as a 2D time-domain forward modeling interactive software package specifically for simulating radargrams via GPR technology since 1989. Based on 2D ray tracing, it is capable of predicting the waveform of microwaves as they interact with model ground structures through reflection, transmission, refraction, and attenuation. The working mechanism is based on the transmission line model and up to ten layers can be simulated with frequencies ranging from 200 to 2000 MHz. The advantage of this software is the inclusion of the antenna design and aperture size in the computations. However, only 2D subsurface scenarios can be modelled. On the other hand, dispersive material and complicated

soil environment with realistic dielectric and geometric properties are not considered in this simulator.

CST is a powerful 3D EM field simulation software. It provides a comprehensive range of time-domain and frequency-domain EM solvers. As the principle of GPR developed for subsurface detection is based on the scattering of EM waves, CST can also be used to model GPR system [67-68]. However, when we conduct the GPR survey for a large subsurface area especially for 3D scenarios, the computation time of CST is very long and the modelling of complicated scenarios such realistic soil environments is difficult to realize. Another weakness of CST is the requirement for the license which is costly.

gprMax is an open-source software for simulating EM wave propagation originally developed in 1996 and continually improved until 2015. It is specially designed for modelling GPR survey covering a wide frequency range via solving Maxwell's equations in 3D via the Finite-Difference Time-Domain (FDTD) method, in which the spatial and temporal derivatives in the finite difference expressions are of a central-difference nature and exhibit second-order accuracy [69]. The code of gprMax is written in Python and friendly to users. The following advanced features have been implemented in the software [65]:

- Scripting in the input file.
- Built-in library of antenna models.
- Anisotropic material modelling.
- Dispersive material modelling.
- Building heterogeneous objects using fractal distributions.
- Building objects with rough surfaces.
- Modelling soils with realistic dielectric and geometric properties.
- Improved perfectly matched layer (PML) performance.

Furthermore, gprMax has implemented the graphics processing unit (GPU) usage in

[66] to accelerate the simulation process. After incorporating GPU utilization for executing the FDTD solver loops, which are the most computationally intensive parts of gprMax, the computation speed has been significantly accelerated. This is beneficial for generating large sets of GPR data in our case.

An example of simulating a 2D subsurface heterogeneous scenario with GPR survey using the gprMax software is shown in Figure 2.14. To model the heterogeneous soil environment with realistic dielectric and geometric properties, a Peplinski mixing model implemented in gprMax is used [70-72]. The properties of the survey scenario are specified in the input scripts as shown in Figure 2.14(a). The definition of each command is explained below the command. The sand fraction, clay fraction, bulk density, and sand particle density of the soil are set to 0.5, 0.5, 2 g/cm^3, and 2.66 g/cm^3, respectively. 20 different materials described by 20 Debye functions over a range of water volumetric fractions from 0.1% to 20% are randomly distributed in the soil. Two objects with different properties are buried in the soil. The usage of all the input commands can be found at http://docs.gprmax.com/en/latest/. The corresponding survey scenario is presented in Figure 2.14(b). The simulated B-scan for this survey scenario is also shown in Figure 2.14(c) but only direct coupling and ground reflections can be observed. Due to high loss of the heterogeneous soil, the reflections from the subsurface objects are too small to be visually seen compared to the direct coupling and ground reflections. After shortening the range of the field strength to [-15, 15] V/m in the visualized B-scan, the hyperbolic patterns reflected from the subsurface objects can be observed as shown in Figure 2.14(d). To make the object signatures visible in the B-scan, GPR data pre-processing algorithms are usually used to remove the strong direct coupling and ground reflections.

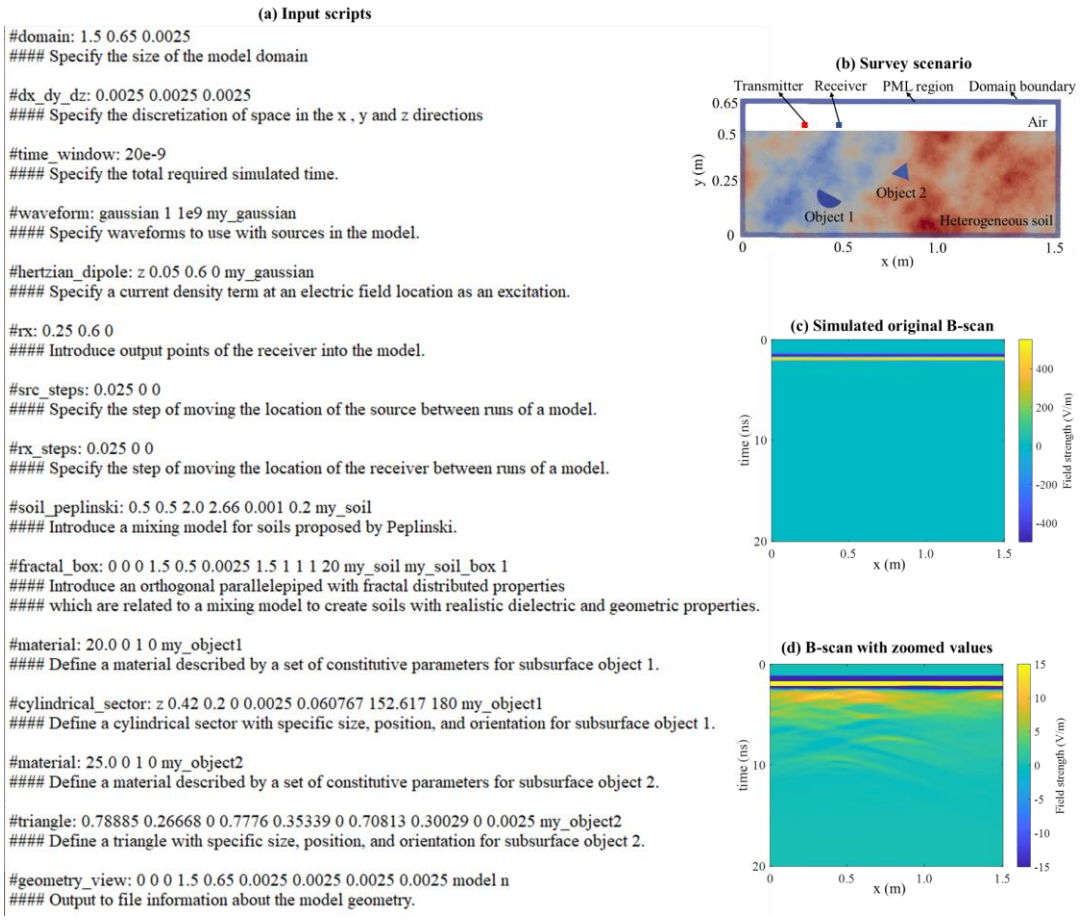


Figure 2.14: An example of the scripts in an input file for simulating a 2D subsurface heterogeneous scenario with GPR survey. (a) The scripts in the input file. (b) The modelled subsurface scenario. (c) The simulated original B-scan. (d) The visualized B-scan with the shortened range of the field strength.

2.3.2 Traditional Forward Solver

The forward full-wave modelling of the GPR system is of great significance for understanding the subsurface scattering mechanisms and interpretation [73]. Furthermore, due to the data-driven characteristics of deep learning-based inversion techniques, large amounts of labelled datasets are required in the training phase as shown in Figure 1.4(a). To this end, it is critical to developing a fast GPR forward solver for data augmentation of deep-learning-based GPR inversion algorithms.

Traditionally, many forward full-wave solvers have been developed so far to characterize the radar signatures of subsurface objects. These solvers are primarily based on the FDTD method [74-75], the method of moments [76], and the finite element time-domain method [77-78]. However, these physics-based traditional

solvers require excessive computational time, especially when their repetitive executions are required by signal processing or machine learning-based inversion algorithms. In particular, the FWI algorithm requires the execution of a forward GPR solver in every iteration of its optimization stage, and an inversion of each GPR data often requires hundreds or even thousands of iterations [35]. Furthermore, the deep learning-based algorithms developed to solve GPR inverse problems require a large dataset for their training and testing stages [16]. For example, over 400,000 GPR B-scans are used to train the deep CNN for imaging subsurface scenarios [58]. This problem exists in our proposed deep learning-based inversion techniques as well. Using a traditional physics-based forward solver, the dataset generation is highly time-consuming.

2.3.3 Deep Learning-Based Forward Solver

To alleviate the computational burden of traditional GPR forward solvers, machine learning-based fast forward solvers have been proposed in [79-82]. For example, the work in [80] explores the relationship between model parameters and first arrival travel time, while the techniques in [81] and [82] form the relationship between the properties of the subsurface scenarios and their corresponding 1D A-scans. All these techniques are proposed to form the parameter-to-parameter relationship between the input parameters related to the GPR scenario and the output parameters, which only yield partial information about the GPR scenario. For example, the A-scan in [82] only depicts the reflected signal at one position, which only contains information in the narrow spatial domain that affects the signal. It cannot capture the reflected signals at different positions, which can potentially contain more object information. Furthermore, the applied subsurface scenarios are simple and limited in the training domain. So far, there exists no study exploiting deep learning algorithms for GPR forward modelling and forming a mapping relationship between the input image of the GPR scenario and the output image (e.g., B-scan) that can provide the complete 2D information of the subsurface scenario.

Chapter 3

2D GPR Data Inversion

This chapter presents the first contribution of this thesis. It provides detailed information on the deep learning-based scheme proposed to solve 2D GPR inverse problem. In particular, this scheme first denoises the GPR B-scans obtained under heterogenous soil conditions and then reconstructs subsurface 2D permittivity distributions. The introduction, methodology, including the 2D two-stage network scheme and training strategy, experiments on numerical and real measurement data, and inversion result analysis are described in detail as follows.

3.1 Introduction

In GPR inverse problems, the properties of the subsurface scenario can be recovered from the EM information in the recorded GPR data. Reconstructing the permittivity distributions of subsurface objects, including constitutive parameters, locations, sizes, and shapes, from the B-scans, has practical significance in non-destructive health monitoring and subsurface utility mapping. As reviewed in Section 2.1, traditional migration algorithms [24-29], microwave tomography approaches [30-33], and 2D FWI algorithms [14-15, 34-36] are limited in three folds: (1) They are not able to restore the objects' constitutive parameters, or (2) they require high computational costs when iteratively constructing the subsurface permittivity maps, and (3) they provide inaccurate results when applied to complex subsurface scenarios. As reviewed in Section 2.2.4, to improve the efficiency and accuracy of restoring subsurface permittivity distributions, DNNs have been investigated to reconstruct 2D subsurface permittivity maps from GPR B-scans [18, 57-59]. However, these existing deep learning-based methods focus on the ideal homogeneous subsurface environments and ignore the interference from clutter and noise due to the heterogeneity of real-world soil environments.

To address these issues, we propose a two-stage DNN, called DMRF-UNet, for reconstructing the permittivity distribution of subsurface objects under heterogeneous

soil conditions. The DMRF-UNet consists of double U-shape neural networks to describe the representation of the GPR inverse problem. The first stage (MRF-UNet1) extracts the hyperbolic signatures reflected from subsurface objects and removes the clutter and noise due to the heterogeneous soil in the input noisy B-scan. The second stage (MRF-UNet2) learns the mapping relationship between the signatures in the denoised B-scan and the permittivity distribution of subsurface objects. The comparative results on numerical and real measurement data demonstrate the superior performance of the proposed method over existing studies.

The main contributions are as follows. First, this is the first time that a deep learning-based framework is proposed for reconstructing the permittivity distributions of subsurface objects under heterogeneous soil conditions. Second, a unique two-stage DNN which outperforms existing works for the reconstruction under heterogeneous soil conditions is carefully designed. In particular, a novel prior denoising network is introduced to extract the objects' signatures and eliminate noise interference under heterogeneous soil conditions. The DNN employs multiple receptive fields to extract multi-scale hyperbolic signatures corresponding to the reflections from multiple objects with various properties in the B-scan. Cascaded small-size convolutional filters are specifically used instead of large-size filters to reduce the computation cost. Besides, the two-stage DNN is trained end-to-end with a combined loss function to balance the training of the two stages and avoid information loss. The proposed network that combines all these components achieves the best performance for reconstructing permittivity distributions under heterogeneous soil conditions.

3.2 Methodology

Let X and Y represent the subsurface scenario and the resulting B-scan obtained after the GPR survey performed on the surface, as shown in Figure 3.1. The forward relationship between the input subsurface scenario and output B-scan is represented by $H(\cdot)$ operator and the GPR forward problem can be described as $Y = H(X)$. In the inverse problem, the aim is to find the inverse transformation $H^{-1}(\cdot)$ from Y to X, which can be described by

$$X = H^{-1}(Y). \tag{3.1}$$

The inverse transformation is highly non-linear, and the inverse problem is ill-posed.

Supervised deep learning provides a data-driven technique to learn $H^{-1}(\cdot)$ to translate the B-scan into the permittivity distribution of subsurface objects. A large set of data is required to train the DNN by minimizing the loss between the predicted and actual permittivity distributions. After the network is well-trained, which represents the optimal $H^{-1}(\cdot)$, the network can automatically predict permittivity distributions from new B-scans.

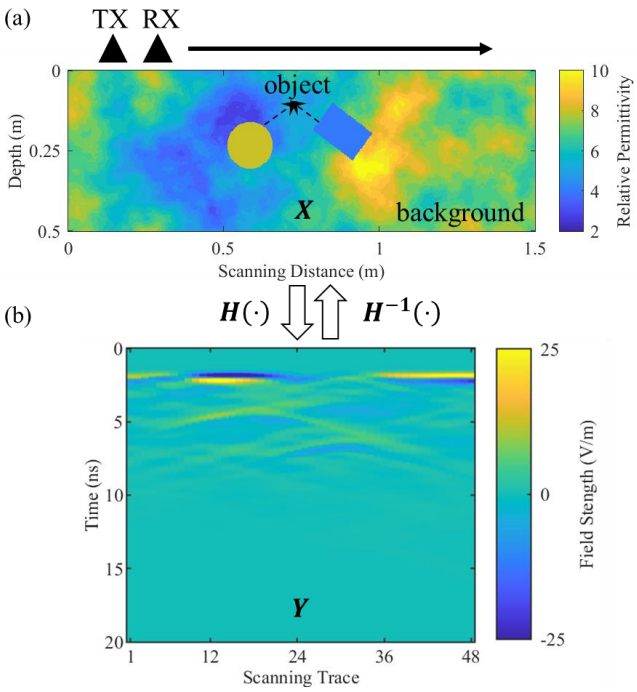


Figure 3.0.1: (a) A typical subsurface scenario with the GPR system under the heterogeneous soil condition. (b) The obtained B-scan image. TX and RX denote the transmitter and the receiver in the GPR survey, respectively. X and Y represent the subsurface scenario and the resulting B-scan obtained after the GPR survey performed on the surface, respectively. $H(\cdot)$ and $H^{-1}(\cdot)$ represent the forward and reverse relationship between X and Y, respectively.

Two main challenges exist to reconstruct the subsurface permittivity distributions from the GPR B-scans under heterogeneous soil environment conditions. First, the heterogeneity of the subsurface environment introduces clutter and noise in the B-scan that interfere with signatures of reflections from the objects. Especially when the object's permittivity is close to that of the surrounding soil, the strength of the field reflected from the object is weak. Thereby, the object's reflection can be easily disguised by environmental clutter. Second, complex subsurface scenarios often yield complicated reflection patterns in the obtained B-scans. When several objects with different properties are close to each other, their corresponding hyperbolic patterns

are overlapped or obscured, which makes it hard to differentiate the objects' signatures and perform the subsequent permittivity distribution reconstruction. To solve these issues, a two-stage DNN is designed to first denoise the B-scan under heterogeneous soil conditions and then reconstruct the permittivity distributions of subsurface objects. We first introduce the working principles of GPR and then describe the details of our proposed scheme for GPR data inversion.

3.2.1 Physical Mechanism of GPR

As described in Section 1.1, GPR detects the subsurface object via transmitting the EM wave into the surface and receiving the reflected signals. It operates based on the physics of EM wave propagation. The EM wave propagation in a medium satisfies the Maxwell's equations, which are expressed in the differential form [1]:

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}, \tag{3.1}$$

$$\nabla \times \boldsymbol{H} = \frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}, \tag{3.2}$$

$$\nabla \cdot \boldsymbol{D} = \rho, \tag{3.3}$$

$$\nabla \cdot \boldsymbol{B} = 0, \tag{3.4}$$

where ∇ is the Hamiltonian, $\boldsymbol{E}$ is the electric field intensity (V/m), $\boldsymbol{B}$ is the magnetic flux density (T), t is time, $\boldsymbol{H}$ is the magnetic field intensity (A/m), $\boldsymbol{D}$ is the electric flux density (C/m^2), $\boldsymbol{J}$ is the current density per unit area (A/m^2), and ρ is the charge intensity per volume area (C/m^3). Dielectric properties including the electric permittivity ε, the magnetic permeability μ, and the electric conductivity σ are incorporated to Maxwell's equations via the following constitutive equations:

$$\boldsymbol{D} = \varepsilon \boldsymbol{E}, \tag{3.5}$$

$$\boldsymbol{B} = \mu \boldsymbol{H}, \tag{3.6}$$

$$\boldsymbol{J} = \sigma \boldsymbol{E}. \tag{3.7}$$

Taking the curl of Equation (3.1) and combining with Equations (3.5) and (3.6), we can obtain the time-domain wave equation:

$$\nabla^2 \boldsymbol{E} - \varepsilon\mu \frac{\partial^2 \boldsymbol{E}}{\partial t^2} - \sigma\mu \frac{\partial \boldsymbol{E}}{\partial t} = 0. \tag{3.8}$$

Using the Fourier transform, we replace $\partial/\partial t$ and $\partial^2/\partial t^2$ with $j\omega$ and $-\omega^2$, respectively, and then Equation (3.8) can be converted into the frequency-domain

equation:

$$\nabla^2 \boldsymbol{E}_\omega + (\varepsilon\mu\omega^2 - j\sigma\mu\omega)\boldsymbol{E}_\omega = 0, \tag{3.9}$$

where ω denotes the angular frequency and $\boldsymbol{E}_\omega$ represents the frequency-domain electric field. Equation (3.9) can be written as

$$\nabla^2 \boldsymbol{E}_\omega - \gamma^2 \boldsymbol{E}_\omega = 0, \tag{3.10}$$

where γ is propagation constant and it is expressed as

$$\gamma = \sqrt{j\omega\mu(\sigma + j\omega\varepsilon)} = \alpha + j\beta, \tag{3.11}$$

where α and β are attenuation constant and phase constant, respectively, and they are given by the following equations [1]:

$$\alpha = \omega \sqrt{\frac{\mu\varepsilon}{2}\left(\sqrt{1 + \frac{\sigma^2}{\omega^2\varepsilon^2}} - 1\right)}, \tag{3.12}$$

$$\beta = \omega \sqrt{\frac{\mu\varepsilon}{2}\left(\sqrt{1 + \frac{\sigma^2}{\omega^2\varepsilon^2}} + 1\right)}. \tag{3.13}$$

Solving Equation (3.10) for monochromatic and plane waves yields

$$\boldsymbol{E}_\omega(\boldsymbol{r}, \omega) = E_0 e^{-\gamma \boldsymbol{r}} = E_0 e^{-\alpha \boldsymbol{r}} e^{-j\beta \boldsymbol{r}}, \tag{3.14}$$

$$\boldsymbol{E}(\boldsymbol{r}, t) = \mathrm{Re}\left(\boldsymbol{E}_\omega(\boldsymbol{r}, \omega) e^{j\omega t}\right) = E_0 e^{-\alpha \boldsymbol{r}} \cos(\omega t - \beta \boldsymbol{r}), \tag{3.15}$$

where $\boldsymbol{r}$ is the displacement vector and E_0 is a constant. Equations (3.14) and (3.15) clearly demonstrate how dielectric properties affect the propagation of electromagnetic waves.

As shown in Figure 3.1, for a GPR system, EM waves are emitted into the ground by the transmitting antenna (TX) and reflected at interfaces or boundaries where there are changes in dielectric properties. When the reflected waves return to the ground surface, they are received by the receiving antenna (RX). When encountered a boundary between medium 1 and medium 2, we can express the incident field $\boldsymbol{E}_i$, reflected field $\boldsymbol{E}_\Gamma$, transmitted field $\boldsymbol{E}_\mathrm{T}$ as

$$\boldsymbol{E}_i = A e^{-j\gamma_1 \boldsymbol{r}}, \tag{3.16}$$

$$\boldsymbol{E}_\Gamma = \Gamma A e^{j\gamma_1 \boldsymbol{r}}, \tag{3.17}$$

$$\boldsymbol{E}_\mathrm{T} = \mathrm{T} A e^{-j\gamma_2 \boldsymbol{r}}, \tag{3.18}$$

where Γ represents the reflection coefficient, T is the transmission coefficient, A is a constant, γ_1 is the propagation constant in medium 1, and γ_2 is the propagation

constant in medium 2. According to the boundary conditions of tangential fields [83], Γ and T are derived as

$$\Gamma = \frac{\sqrt{\mu_2/\varepsilon_2} - \sqrt{\mu_1/\varepsilon_1}}{\sqrt{\mu_2/\varepsilon_2} + \sqrt{\mu_1/\varepsilon_1}}, \tag{3.19}$$

$$\mathrm{T} = \frac{2\sqrt{\mu_2/\varepsilon_2}}{\sqrt{\mu_2/\varepsilon_2} + \sqrt{\mu_1/\varepsilon_1}}, \tag{3.20}$$

where ε_1, ε_2, μ_1, and μ_2 are the permittivity of medium 1, permittivity of medium 2, permeability of medium 1, and permeability of medium 2, respectively. Considering non-magnetic materials, Equations (3.19) and (3.20) can be further expressed as

$$\Gamma = \frac{\sqrt{\varepsilon_1} - \sqrt{\varepsilon_2}}{\sqrt{\varepsilon_1} + \sqrt{\varepsilon_2}}, \tag{3.21}$$

$$\mathrm{T} = \frac{2\sqrt{\varepsilon_1}}{\sqrt{\varepsilon_1} + \sqrt{\varepsilon_2}}, \tag{3.22}$$

Equations (3.21) and (3.22) intuitively show that the reflection coefficient and the transmission coefficient are highly related to the spatial variations of the electric permittivity. For the subsurface environment in a GPR survey, the reflected field strength is highly related to the variation between the permittivity of the target object and the permittivity of the background medium. It should be noted that the permittivity discussed in this thesis is the real part of relative permittivity $\varepsilon_\mathrm{r} = \varepsilon/\varepsilon_0$, where ε_0 is the permittivity of free space.

3.2.2 MRF-UNet1: Signature Extraction

The whole architecture of our proposed two-stage network DMRF-UNet for reconstructing the subsurface permittivity distribution from the GPR B-scan is shown in Figure 3.2. We first introduce the details of the first-stage network for extracting the signatures in the noisy B-scan.

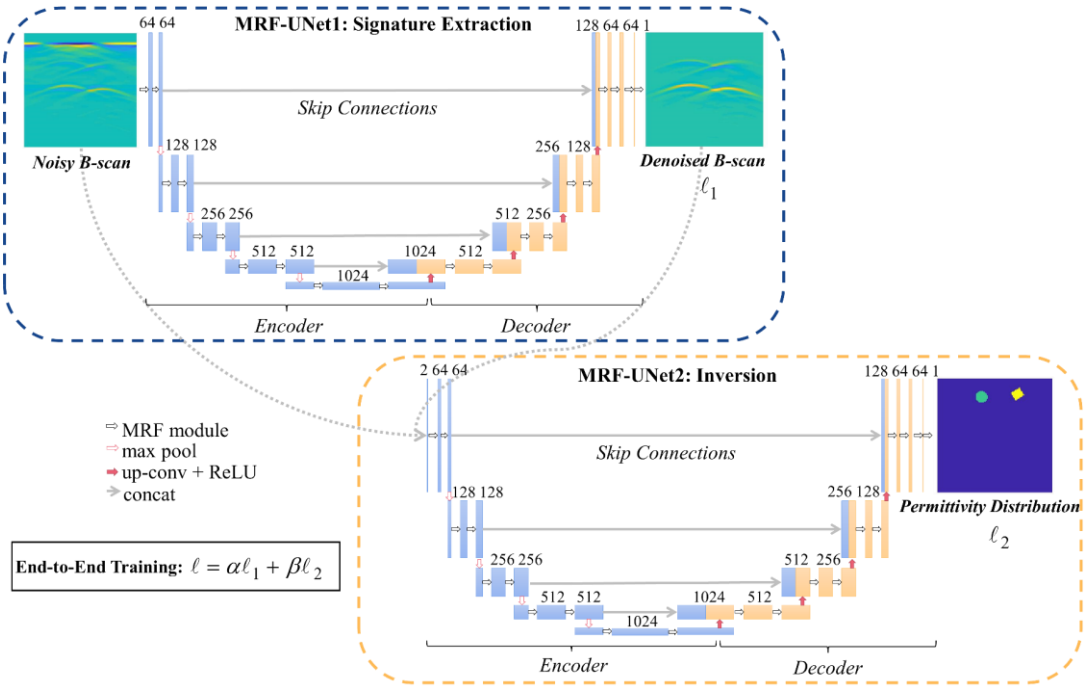


Figure 3.2: The structure of the proposed DMRF-UNet. "MRF module", "max pool", "concat", "up-conv", "conv", and "ReLU" represent the multi-receptive-field module, max-pooling layer, concatenation, up-convolutional layer, convolutional layer, and rectified linear unit activation, respectively. The numbers above the rectangles mean the channels of feature maps. ℓ_1 and ℓ_2 represent the loss functions of MRF-UNet1 and MRF-UNet2. ℓ is the combined loss for the end-to-end training.

Compared with the homogeneous environment, clutter and noise are more pronounced in the B-scans under heterogeneous soil conditions, such as the input noisy B-scan shown in Figure 3.2. It is imperative to first denoise the noisy image and extract the signatures corresponding to the reflections from subsurface objects to improve the quality of B-scans for further analysis. Based on the blind source separation principle in signal processing [84], it is assumed that the observed GPR data (Y) is a mixture of the clutter (Y_c) and the target signal (Y_t) [85], as expressed as

$$Y = Y_c + Y_t. \tag{3.23}$$

Signature extraction aims to remove the clutter component Y_c in the obtained noisy B-scan and to provide a denoised image that only includes the target component Y_t.

To achieve that, a U-shape CNN with MRF modules, namely MRF-UNet1 in Figure 3.2, is designed. The noisy B-scan is obtained by pre-processing the original B-scan by the mean subtraction technique. MRF-UNet1 takes the noisy B-scan Y as input,

extracts key features of the object reflection from Y, suppresses the environmental clutter Y_c, and outputs the denoised B-scan with only object reflections Y_t. The network consists of an encoder and a decoder with skip connections. The encoder is made up of five repeated applications of two MRF modules and one 2×2 max-pooling layer with the stride of 2×2. The decoder includes four repeated applications of one up-convolution layer and two MRF modules. A 2×2 up-sampling layer is combined with a 2×2 convolutional layer as one up-convolution layer. The channels of the convolutional layers in the encoder and decoder are set as [64, 128, 256, 512, 1024] and [512, 256, 128, 64], respectively. To compensate for the information loss in the down-sampling process of the encoder, four skip connections concatenate the feature maps from the up-convolutional layers in the decoder with the corresponding feature maps from the encoder. At the final stage, a 1×1 convolutional layer followed by the ReLU activation outputs the denoised B-scan, which only has hyperbolic signatures of the reflections from objects.

In the training phase, the ground truth of the denoised image is generated by subtracting the simulated B-scan under heterogeneous soil conditions without any objects from the B-scan under heterogeneous soil conditions with the objects. The denoised images only have the hyperbolic signatures from the objects, as shown in the output image of the MRF-UNet1 in Figure 3.2. Note that the ground truth of the prior denoised image is only required in the training phase. The well-trained network will automatically predict the denoised B-scans in the testing phase.

3.2.3 MRF-UNet2: Inversion

After extracting the object-related signatures and removing the soil-related noise, the MRF-UNet2 is introduced to accomplish the task of inversion, which translates the EM information in the B-scans into the permittivity distributions of subsurface objects. The configuration of MRF-UNet2 is shown in Figure 3.2. The noisy B-scan and the denoised B-scan from MRF-UNet1 are concatenated as a two-channel input of the MRF-UNet2. This arrangement is made to allow the MRF-UNet2 to concentrate on the object-related signatures from the predicted denoised B-scan while avoiding loss of information in the B-scan due to the denoising process of MRF-UNet1. In this way, the reconstruction capability of MRF-UNet2 can be enhanced by learning more informative features from the hybrid B-scans and building a more

accurate mapping relationship from B-scans to subsurface permittivity distributions in the training process. Furthermore, the skip connection between the input layer and the medium layer of the network provides implicit deeper supervision for the training, resulting in improved prediction performance [61, 86]. Using the same structure as MRF-UNet1, MRF-UNet2 is composed of an encoder and a decoder with skip connections, while the input is the hybrid two-channel B-scan and the output is the subsurface permittivity distribution. At the final stage, a 1×1 convolutional layer followed by the exponential linear unit activation outputs the permittivity distribution of subsurface objects. Other parameter settings in MRF-UNet2 are the same as those in MRF-UNet1.

3.2.4 Multi-Receptive-Field Module

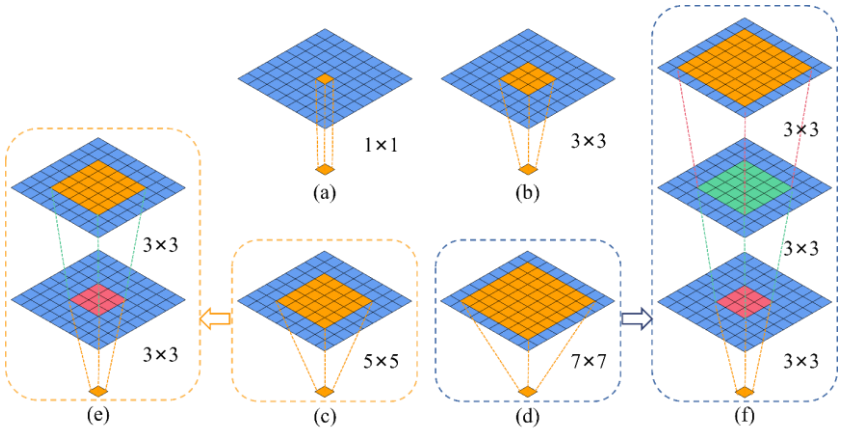
A GPR B-scan usually includes multiple hyperbolic patterns at different spatial locations due to the randomness of the shape, location, and property of the subsurface object. To effectively extract discriminative features in different spatial scales, inspired by the Inception module in [87-89], a specific MRF module is designed in the convolutions. The receptive field is theoretically defined as the region in the input image that a particular CNN's feature is looking at, which means how large the pixels in the high-level feature map are affected by the input image [90]. Large receptive fields capture more global features with higher-level semantic information, while small receptive fields perceive the features that focus on the local details. The receptive field size is computed by [91]

$$r_\eta = r_{\eta-1} + \left(k_\eta - 1\right) \times \prod_{l=1}^{\eta-1} s_l, \tag{3.24}$$

where r_η is the receptive field of the ηth layer, $r_{\eta-1}$ is the receptive field of the $(\eta - 1)$th layer, k_η is the kernel size and s_l is the stride in the convolutions. If s_l is set as 1 in every convolutional layer, the receptive field is determined by the kernel size of the current layer k_η and the receptive field of the former layer $r_{\eta-1}$. Thus, four kernel sizes, 1×1, 3×3, 5×5, and 7×7, as shown in Figures 3.3(a)-(d), are designed to achieve four different receptive fields, 1×1, 3×3, 5×5, and 7×7, to extract multi-scale features in the B-scan. However, the computational cost increases with a larger kernel size. To alleviate the computational burden, the mini-network replacement scheme [88] is

used, which replaces the 5×5 convolution with two cascaded 3×3 convolutional layers [Figure 3.3(e)] and replaces the 7×7 convolution with three cascaded 3×3 convolutional layers [Figure 3.3(f)]. The receptive field after the replacement remains unchanged based on Equation (3.3). Assuming the channels of the feature map is C, the required numbers of trainable parameters in the 5×5 and 7×7 convolution are $25C^2$ and $49C^2$, respectively. After applying the mini-network replacement, the numbers of parameters are reduced to $18C^2$ and $27C^2$, respectively, demonstrating the effectiveness of the replacement scheme in alleviating the computational burden. Moreover, the depth of the network increases and the network capacity and complexity are enhanced with the mini-network replacement scheme.

The detailed structure of one MRF module is presented in Figure 3.3(g). Firstly, convolutions with four sizes of receptive fields: 1×1, 3×3, 5×5 (two cascaded 3×3 convolutional layers), and 7×7 (three cascaded 3×3 convolutional layers), are conducted. Each convolutional layer is followed by the ReLU activation. The stride of each convolutional layers is 1×1. Then the obtained four feature maps are concatenated, followed by a 3×3 convolutional layer. The final number of channels becomes four times the previous. The MRF module is implemented in both the MRF-UNet1 for signature extraction and the MRF-UNet2 for inversion.



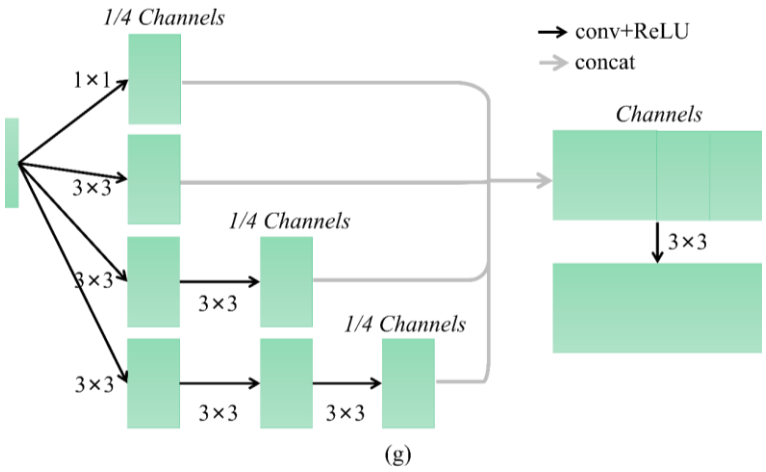


Figure 3.3: Four receptive fields from four different kernel sizes: (a) 1×1, (b) 3×3, (c) 5×5, and (d) 7×7. (e) The 5×5 convolution in (c) can be replaced by two cascaded 3×3 convolutional layers. (f) The 7×7 convolution in (d) can be replaced by three cascaded 3×3 convolutional layers. (g) The structure of one MRF module. The green rectangles represent the feature maps. "conv", "ReLU", and "concat" represent the convolutional layer, rectified linear unit activation, and concatenation, respectively.

3.2.5 End-to-End Training

The losses of MRF-UNet1 and MRF-UNet2, ℓ_1 and ℓ_2, are combined in the loss function to implement end-to-end training of the proposed framework. Both the ℓ_1 and ℓ_2 use the mean square error (MSE) loss. The combined loss ℓ is defined as:

$$\ell = \alpha\ell_1 + \beta\ell_2$$

$$= \alpha \frac{1}{H_1 \times W_1} \sum_{i_1,j_1} \left(Y_{t_{i_1,j_1}} - \hat{Y}_{t_{i_1,j_1}}\right)^2 + \beta \frac{1}{H_2 \times W_2} \sum_{i_2,j_2} \left(X_{i_2,j_2} - \hat{X}_{i_2,j_2}\right)^2, \qquad (3.25)$$

where ℓ is the combined loss. ℓ_1 and ℓ_2 are the losses of MRF-UNet1 and MRF-UNet2, respectively. When ℓ_1 and ℓ_2 are different in order of magnitude in the training process, the training will be biased towards the sub-network with a larger loss. To avoid this issue, the loss weights α and β are used to balance the training of MRF-UNet1 and MRF-UNet2. The values of α and β are determined by the quantitative difference between ℓ_1 and ℓ_2. H_1 and W_1 are the dimensions of the denoised B-scan, while i_1 and j_1 are the indices. H_2 and W_2 are the dimensions of the permittivity distribution, while i_2 and j_2 are the indices. Y_t and $\hat{Y}_t$ are the predicted

result and the ground truth of the denoised B-scan. X and $\hat{X}$ are the predicted result and the ground truth of the permittivity distribution.

3.3 Numerical Experiments

3.3.1 Numerical Dataset Preparation

To train and test the proposed DMRF-UNet, a large diverse set of data under heterogeneous soil conditions is generated using a GPU-based open-source software gprMax [65-66]. Each set of data includes three images, the input noisy B-scan, the denoised B-scan, and the permittivity distribution of the subsurface environment with buried objects. To obtain the noisy B-scans under the heterogeneous soil environment with realistic dielectric and geometric properties, a Peplinski mixing model implemented in gprMax is used [70-72]. The properties of soil are set as sand fraction 0.5, clay fraction 0.5, bulk density 2 g/cm^3, sand particle density of 2.66 g/cm^3. The fractal dimension is set as 1.5 and the weights of the fractals along the x, y, and z axes are the same. 20 different materials over a range of water volumetric fractions from 0.1% to 20%, described by 20 Debye functions, are randomly distributed in the soil. The relative permittivity and conductivity of the soil vary in [3.82, 9.99] and [0.01, 0.07], respectively. To increase the diversity of the heterogeneous soil, ten random distributions are generated to model 10 heterogeneous scenarios as shown in Figure 3.4.

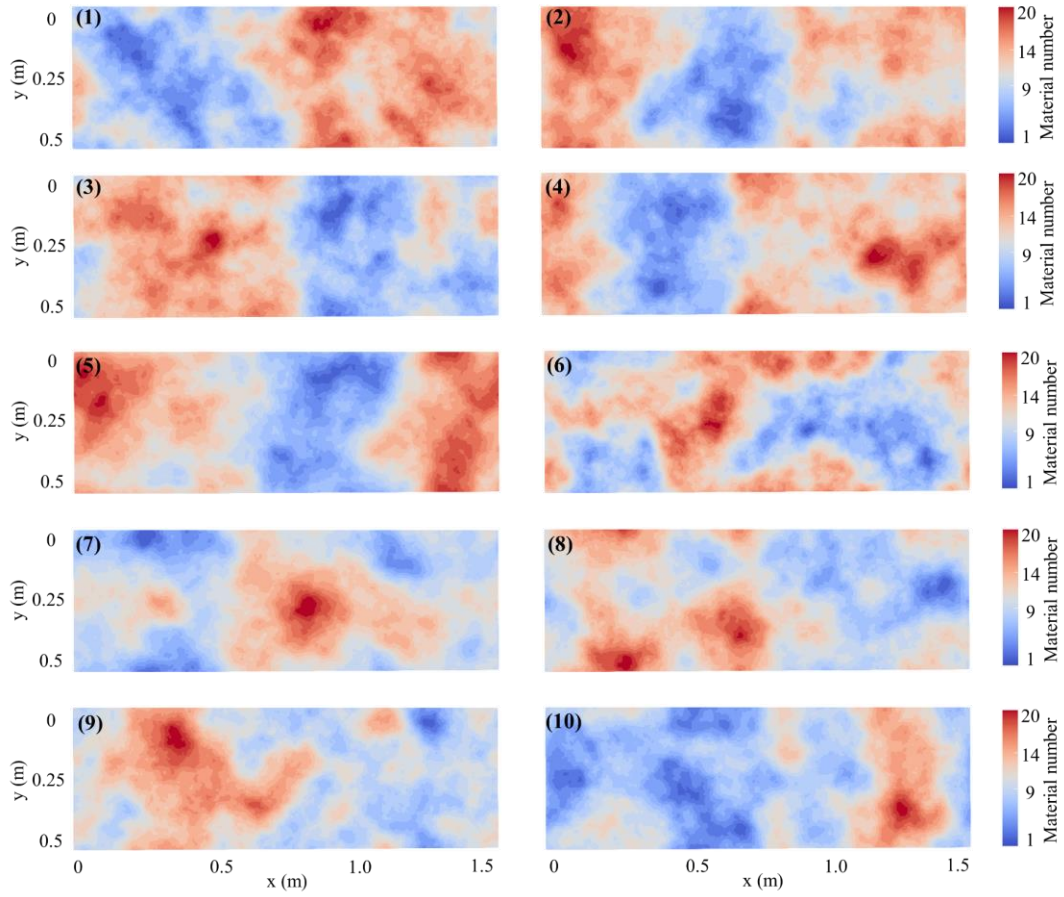


Figure 3.4: Ten heterogeneous soil models using ten random distributions with fixed 20 materials.

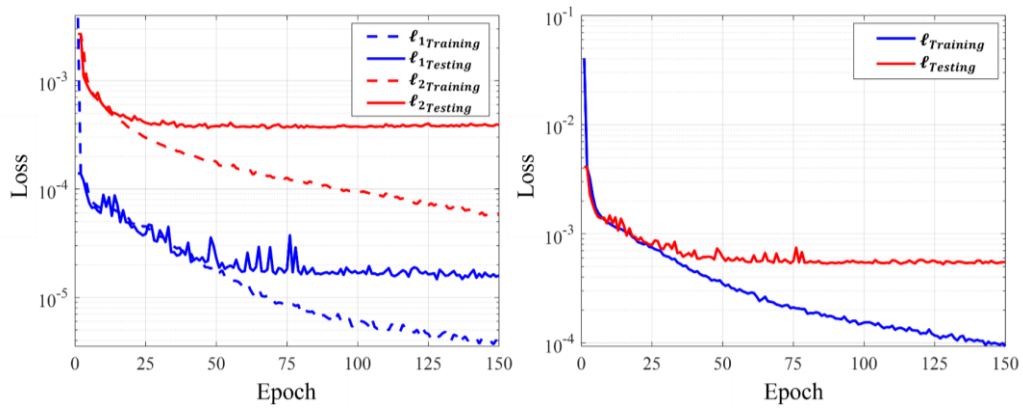
One example of heterogeneous soil with two buried objects is shown in Figure 3.1(a). The size of the 2D soil domain is 1.5×0.5 m^2 and the resolution is 0.0025×0.0025 m^2. The time window is set as 20 ns. A Gaussian waveform with an amplitude of 1 A and a center frequency of 1 GHz is excited. A Hertzian dipole (TX) is used to transmit the signals and a probe (RX), 20 cm away from the dipole, is used to receive the signals. The distance between the TX/RX and the soil surface is 10 cm. In a common-offset mode, the TX/RX are moving along one straight scanning trajectory to obtain a B-scan. The scanning step is 2.5 cm. The shape of the buried object is randomly selected from circles, semi-circles, triangles, and rectangles. The object's permittivity is randomly selected from [2, 32]. For the circle and semi-circle, the radius is randomly selected from [0.05, 0.08] m. For the triangle, the distance from the three vertices to the center is randomly selected from [0.05, 0.08] m. The x and y positions of the center are randomly selected from [0.25, 1.25] m and [0.25, 0.4] m, respectively. For the rectangle, the x and y positions of the left bottom vertex are selected from [0.5, 1] m and [0.25, 0.3] m. The width and length are chosen from

[0.04, 0.06] m and [0.12, 0.16] m. The orientations of the semi-circle, triangle, and rectangle are selected from [0, 360] degrees. 8,000 one-object scenarios and 10,000 two-object scenarios are simulated to obtain the B-scans.

To obtain the denoised B-scans as the prior label of the MRF-UNet1, 10 heterogeneous environments without any objects are used as the simulation scenarios to obtain the soil-only B-scan with only background clutter. After that, the soil-only B-scans are subtracted from the noisy B-scans to create the object-only (denoised) B-scans. To generate permittivity distributions as the second label, we set the soil area as 0 and the object region as the object's permittivity. The dataset, including the noisy B-scans, the denoised B-scans, and the corresponding permittivity distributions, is used for training and testing purposes.

3.3.2 Implementation Details

After obtaining the dataset, 16200 sets of data (7200 one-object and 9000 two-object scenarios) are used for training while the rest 1800 sets of data (800 one-object and 1000 two-object scenarios) are for testing. The original B-scans are pre-processed by mean subtraction. Then all the images are normalized to [0, 1] and resized to 128×128. The proposed DMRF-UNet is implemented on TensorFlow [92]. Note that the whole network has one input (the noisy B-scan) and two outputs (the denoised B-scan and the permittivity distribution). The end-to-end training is employed based on the loss function (3.25). The Adam optimizer [93] is used to perform the optimization. The learning rate is set as 0.0001. The network is trained for 150 epochs and the best model with the lowest testing loss is saved. Figure 3.5(a) shows the training and testing loss curves of ℓ_1 and ℓ_2 of the two sub-networks, respectively. In the training loss curves, ℓ_2 is always about 10 times larger than ℓ_1. To balance the training of MRF-UNet1 and MRF-UNet2, α and β are set as 10 and 1, respectively. Figure 3.5(b) presents the combined loss curves of ℓ with good convergence.



(a) (b)

Figure 3.5: The training and testing loss curves of (a) ℓ_1, ℓ_2, and (b) ℓ, where ℓ_1, ℓ_2, and ℓ represent the loss of the first-stage network MRF-UNet1, the loss of the second-stage network MRF-UNet2, and the total loss of the network DMRF-UNet that combines the first stage and the second stage, respectively.

Four metrics are applied to the testing data to quantitatively evaluate the inversion performance of the proposed scheme. The structural similarity (SSIM) [94] is adopted to evaluate the similarity of the predicted result P (Y_t or X) and the ground truth T ($\hat{Y}_t$ or $\hat{X}$) (the larger, the better):

$$SSIM(P,T) = \frac{(2\mu_P\mu_T + c_1)(2\sigma_{PT} + c_2)}{(\mu_P^2 + \mu_T^2 + c_1)(\sigma_P^2 + \sigma_T^2 + c_2)}, \tag{3.26}$$

where μ_P and μ_T are the means of P and T. σ_T and σ_P are the variances of T and P, respectively. σ_{PT} is the covariance of P and T. $c_1 = (0.01R)^2$ and $c_2 = (0.03R)^2$ are two variables where R is the dynamic range of the image. The MSE, mean absolute error (MAE), and mean relative error (MRE) evaluate the errors between P and T (the smaller, the better):

$$MSE(P,T) = \frac{1}{H \times W}\sum_{i,j}\left(P_{i,j} - T_{i,j}\right)^2, \tag{3.27}$$

$$MAE(P,T) = \frac{1}{H \times W}\sum_{i,j}\left|P_{i,j} - T_{i,j}\right|, \tag{3.28}$$

$$MRE(P,T) = \frac{1}{H \times W}\frac{\sum_{i,j}\left|P_{i,j} - T_{i,j}\right|}{max\left(\left|T_{i,j}\right|\right)} \times 100\%. \tag{3.29}$$

3.3.3 Inversion Results and Comparative Study

1) Comparison with Other Deep Learning-based Methods

Table 3.1: Comparison with deep learning-based studies on evaluation metrics of the testing data

Network	SSIM (↑)	MSE (↓)	MAE (↓)	MRE (%) (↓)
PINet [59]	0.9617	0.8552	0.0563	0.2987
Enc-Dec [61]	0.9794	0.5222	0.0428	0.2158
U-Net [61]	0.9803	0.4968	0.0399	0.2039
SMRF-UNet	0.9823	0.4252	0.0356	0.1858
DMRF-UNet	**0.9845**	**0.3867**	**0.0317**	**0.1642**

The testing performance of the proposed DMRF-UNet is compared with four deep learning-based models: (1) PINet [59], which is state-of-the-art GPR inversion work based on deep learning, (2) Enc-Dec (Note: To control the variables in the comparison, we adopt the encoder-decoder structure in U-Net [61] but remove the skip connections), (3) U-Net [61], and (4) the single MRF-UNet (SMRF-UNet) without the MRF-UNet1. For all these models, the permittivity distribution of subsurface objects is predicted from the noisy B-scan directly without the inclusion of the denoising sub-network. For the models (1)-(3), the convolutions are regular ones without MRF modules. After inversely normalizing the output denoised B-scans and permittivity distributions to [-50, 75] V/m and [0, 32], respectively, the comparison on the average evaluation metrics of the testing samples is shown in Table 3.1. As shown, the DMRF-UNet achieves the best inversion performance on all the metrics and the SMRF-UNet is the second best compared to the PINet, Enc-Dec, and U-Net. This is because the networks (1)-(3) employ single-receptive-field convolutions, which leads to a weaker feature extraction capability when facing scenarios with multiple objects buried in the heterogenous soil. On the other hand, all the networks (1)-(4) extract features only from the noisy B-scan. The clutter and noise from the heterogeneous soil make the identification of the objects' hyperbolic signatures difficult, which is also verified by the SMRF-UNet's performance degradation compared to the DMRF-UNet's. However, the DMRF-UNet includes MRF modules to extract multi-scale signatures, the denoising sub-network to first extract the objects' hyperbolic signatures, and the inversion sub-network to perform

the precise reconstruction of the permittivity distribution. In addition, for the denoising performance of MRF-UNet1, the SSIM is as high as 0.9981 and the MSE, MAE, and MRE are as low as 0.0674, 0.1212, and 1.3869%, respectively, demonstrating the high accuracy of the first-stage signature extraction.

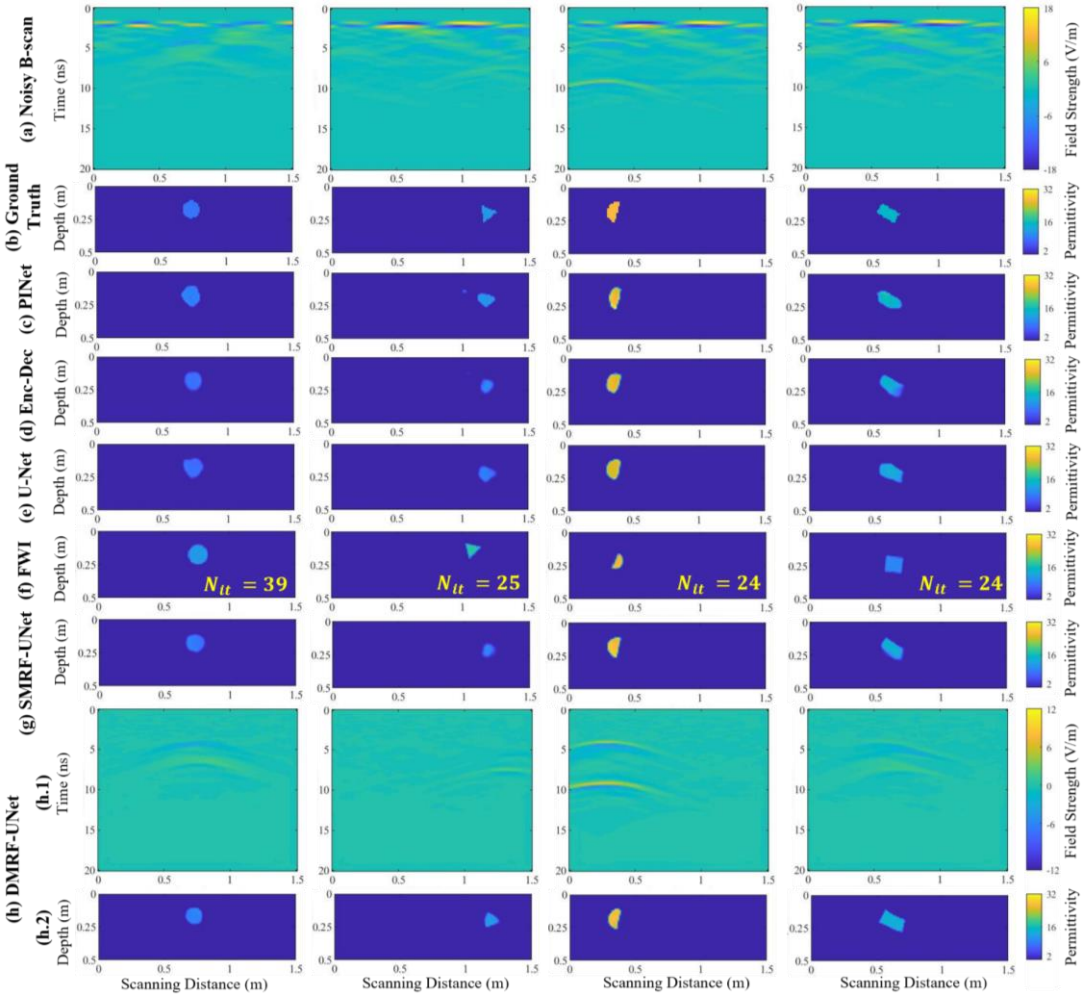


Figure 3.6: Inversion result comparison when one object is buried. (a) Input noisy B-scans. (b) Ground truths of the permittivity distribution of subsurface objects. (c) Predicted permittivity distributions from the PINet. (d) Predicted permittivity distributions from the Enc-Dec. (e) Predicted permittivity distributions from the U-Net. (f) Reconstructed permittivity distributions using the FWI algorithm. The N_{it} stands for the number of iterations required until the convergence of the optimization process. (g) Predicted permittivity distributions from the SMRF-UNet. (h) Predicted results from the proposed DMRF-UNet including (h.1) denoised B-scans from MRF-UNet1, and (h.2) permittivity distributions from MRF-UNet2. Note that the soil area (in the darkest blue) has a value of 0. The "Permittivity" and "Field Strength" represent the relative permittivity of the subsurface objects and the electric field strength, respectively.

The comparison among the predicted permittivity distributions with one object buried

in the heterogeneous soil is shown in Figure 3.6. Figures 3.6(a) and (b) show four examples of the input noisy B-scans and ground truths of permittivity distributions of the subsurface object, respectively. Figures 3.6(c), (d), (e), and (g) show the predicted permittivity distributions from the PINet, Enc-Dec, U-Net, and SMRF-UNet, respectively. Figure 3.6(h) presents the predicted results from the proposed DMRF-UNet, including the predicted denoised B-scan from MRF-UNet1 in Figure 3.6(h.1), and the predicted permittivity distributions from MRF-UNet2 in Figure 3.6(h.2). By comparing all these predicted results, the reconstructed permittivity distributions of the subsurface objects from the DMRF-UNet are more accurate than other works; especially the shapes of the objects are much closer to the ground truths.

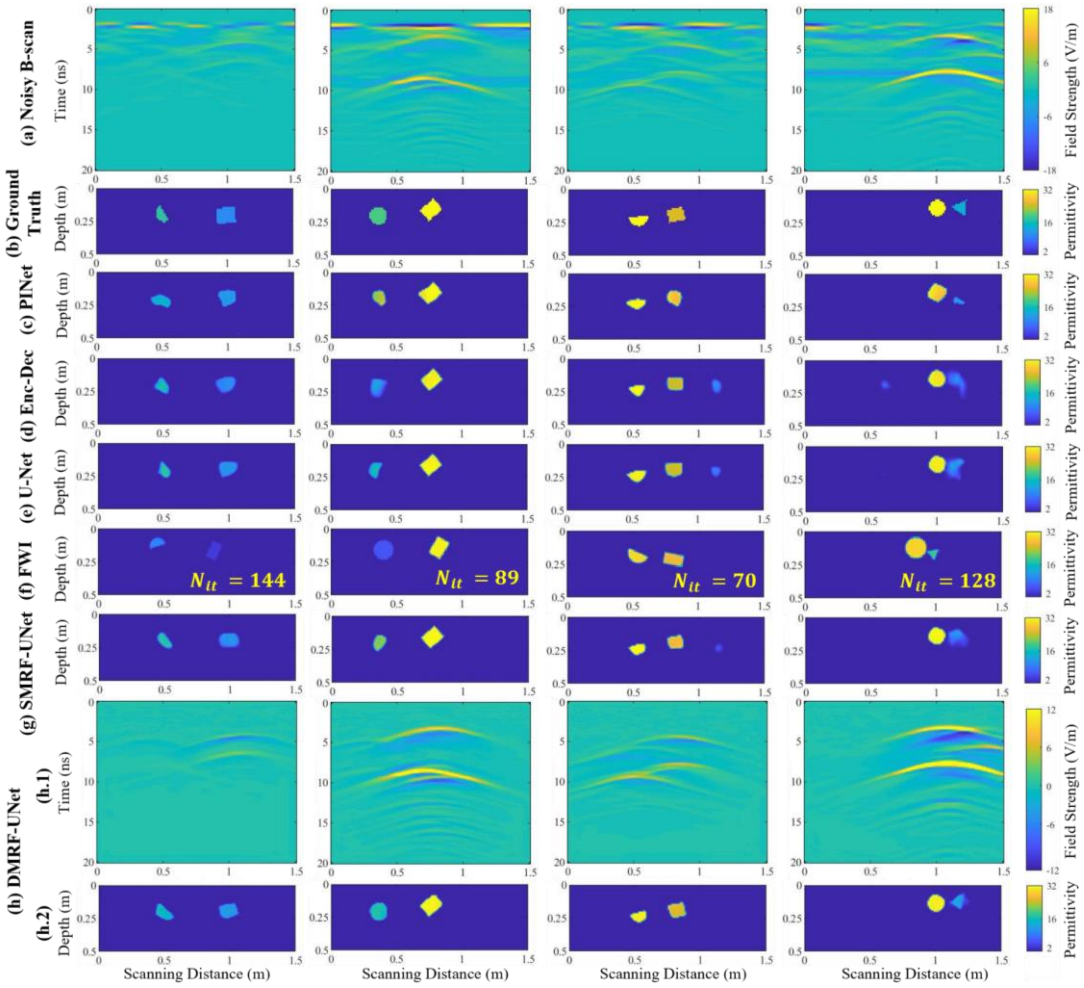


Figure 3.7: Inversion result comparison when two separated objects are buried. (a) Input noisy B-scans. (b) Ground truths of the permittivity distribution of the subsurface objects. (c) Predicted permittivity distributions from PINet. (d) Predicted permittivity distributions from the Enc-Dec. (e) Predicted permittivity distributions from the U-Net. (f) Reconstructed permittivity distributions using the FWI algorithm. The N_{it} stands for the number of iterations required until the convergence of the

optimization process. (g) Predicted permittivity distributions from the SMRF-UNet. (h) Predicted results from the proposed DMRF-UNet.

Figure 3.7 shows the inversion results when two separated objects are buried. Two-object scenarios are more challenging to reconstruct than the one-object scenarios as the hyperbolic patterns from different objects may interfere with or disguise each other. As shown in Figures 3.7(c)-(e), the networks in the existing studies cannot recover the permittivity distributions in these complex scenarios. Some of them even misinterpret the noise as the reflected signal from an extra object. As shown in Figure 3.7(g), the SMRF-UNet recovers more accurate shapes and permittivity values of the objects than the previous networks, but still recognizes some noise as the target signal. Nevertheless, the proposed DMRF-UNet successfully extracts the main signatures [Figure 3.7(h.1)] and accurately reconstructs the permittivity distributions of two separated objects [Figure 3.7(h.2)]. Comparing with the results obtained using PINet, Enc-Dec, U-Net, and SMRF-UNet, the predicted permittivity distributions from the proposed scheme, shown in Figure 3.7(h), are more precise regarding the shapes, sizes, positions, and permittivity values of the subsurface objects.

Figure 3.8 compares the inversion results when two subsurface objects are interfaced. In these cases, the hyperbolic patterns corresponding to reflections from two objects are highly interleaved, making the reconstruction of the permittivity distribution even more difficult. As shown in Figures 3.8(c)-(e), the PINet, Enc-Dec, and U-Net fail to differentiate two objects, and the predicted permittivity values greatly deviate from the ground truth. As shown in Figure 3.8(g), the SMRF-UNet can roughly reconstruct the profiles of two interfaced objects, but the boundaries are quite blurred. On the contrary, the proposed network is capable of accurately reconstructing the permittivity distributions of the interfaced objects, as shown in Figure 3.8(h). These results prove the effectiveness of the proposed scheme in accomplishing the inversion task with a higher resolution.

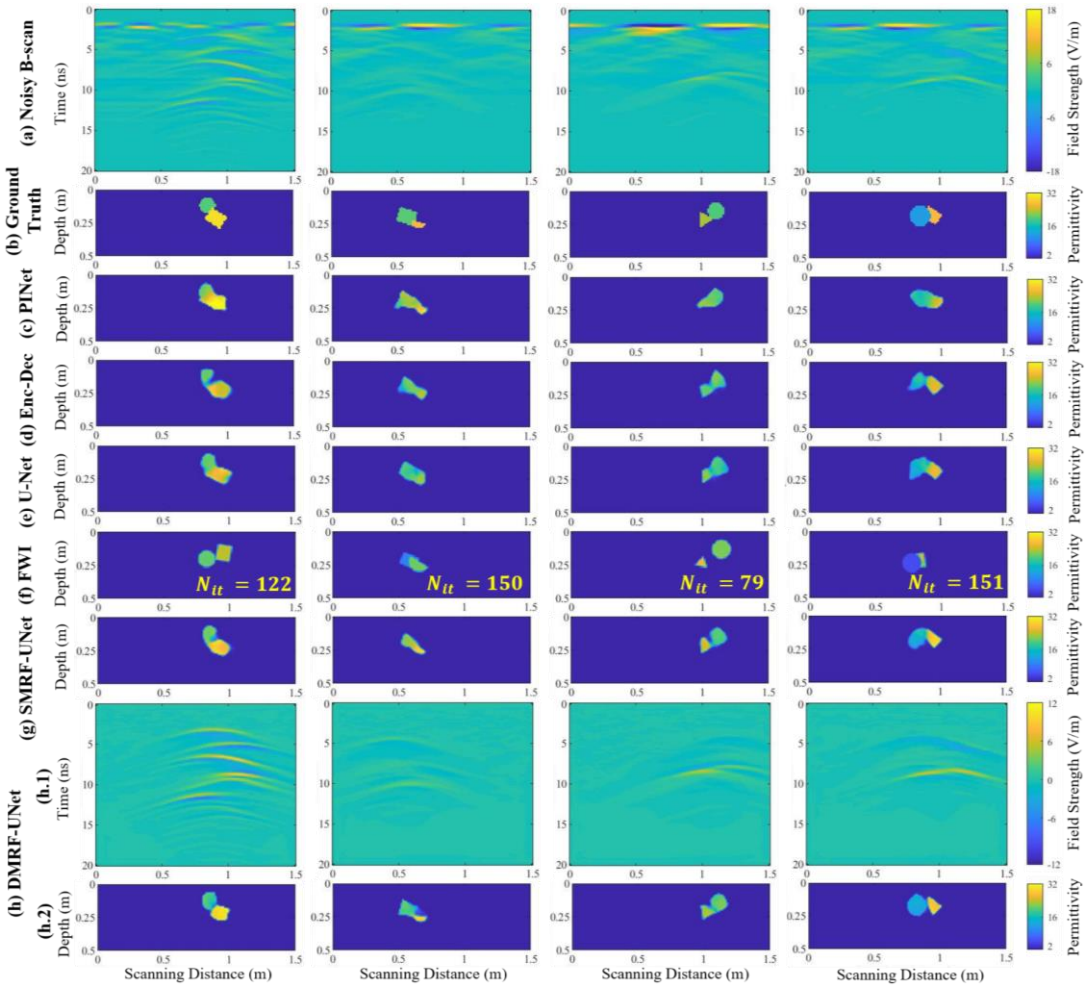


Figure 3.8: Inversion result comparison when two interfaced objects are buried. (a) Input noisy B-scans. (b) Ground truths of the permittivity distribution of the subsurface objects. (c) Predicted permittivity distributions from PINet. (d) Predicted permittivity distributions from the Enc-Dec. (e) Predicted permittivity distributions from the U-Net. (f) Reconstructed permittivity distributions using the FWI algorithm. The N_{it} stands for the number of iterations required until the convergence of the optimization process. (g) Predicted permittivity distributions from the SMRF-UNet. (h) Predicted results from the proposed DMRF-UNet.

2) Comparison with Conventional FWI Algorithm

To verify the superiority of the proposed scheme over traditional methods in reconstructing subsurface permittivity distributions from GPR B-scans, a comparative study with the widely used physics-driven FWI algorithm is performed. To apply the FWI algorithm to our cases, the 2D FDTD gprMax simulator [65-66] is used for forward modeling in each iteration [95-96]. Simulated annealing optimization [97] is performed to minimize the MSE between the observed B-scan and the synthetic B-scan via updating the trial properties of subsurface objects. With

the optimal object properties, the permittivity distribution of subsurface objects can be reconstructed for each observed B-scan. It should be noted that the aim of the FWI algorithm is to iteratively find the optimal permittivity distribution for a given B-scan, whereas the aim of deep learning-based methods is to find the optimal mapping relationship between B-scans and permittivity distributions.

Figures 3.6-3.8(f) present the inversion results using the FWI algorithm for four one-object, two-separated-object, and two-interfaced-object scenarios, respectively. As shown in Figure 3.6(f), FWI can roughly restore the permittivity distribution of one object, while the predicted sizes and shapes/orientations of the objects deviate from the ground truths. The number of iterations, N_{it}, required for convergence varies from 24 to 39. For two-separated-object scenarios shown in Figure 3.7(f), the objects can be clearly observed in the images reconstructed using FWI, but some objects' permittivity values and orientations are different from those in the ground truth images. For the two-interfaced-object cases shown in Figure 3.8(f), the reconstructed profiles are far from their corresponding ground truths, especially regarding the objects' positions and orientations. As the updated variables in the optimization process for two-object scenarios are more than those for one-object scenarios, the required iteration number increases significantly, ranging from 70 to 151. Instead, the permittivity distributions reconstructed by the SMRF-UNet and DMRF-UNet are much closer to the ground truths, especially the DMRF-UNet that includes the denoising sub-network performs the best regarding the shapes, orientations, sizes, positions, and permittivity values of the objects.

To quantitatively compare the performance of the FWI, SMRF-UNet, and DMRF-UNet, the evaluation metrics of the three types of scenarios shown in Figures 3.6-3.8 are listed in Table 3.2. As shown, all the FWI, SMRF-UNet, and DMRF-UNet perform better in one-object scenarios than in two-separated-object and two-interfaced-object scenarios, as the permittivity reconstruction of two objects is more challenging than that of one object. In each scenario type, the MSE, MAE and MRE values of DMRF-UNet are the lowest, and the SSIM is the highest, and those of SMRF-UNet are the second best, while the FWI performs the worst.

Table 3.2: Comparison with FWI algorithm on evaluation metrics of the four sets of testing data shown in Figures 3.6-3.8.

Scenario Type		One Object	Two Separated Objects	Two Interfaced Objects	Ave.
MSE (↓)	FWI	1.5693	7.0984	3.8661	4.7119
	SMRF-UNet	0.2225	0.8116	0.9687	0.6676
	DMRF-UNet	**0.1804**	**0.5544**	**0.8579**	**0.5298**
MAE (↓)	FWI	0.0950	0.3044	0.2089	0.2028
	SMRF-UNet	0.0239	0.0675	0.0714	0.0543
	DMRF-UNet	**0.0196**	**0.0596**	**0.0571**	**0.0412**
MRE (%) (↓)	FWI	0.6598	1.0541	0.7721	0.8287
	SMRF-UNet	0.1736	0.2139	0.2656	0.2177
	DMRF-UNet	**0.1558**	**0.1847**	**0.2116**	**0.1840**
SSIM (↑)	FWI	0.9709	0.9379	0.9590	0.9559
	SMRF-UNet	0.9868	0.9693	0.9799	0.9787
	DMRF-UNet	**0.9875**	**0.9713**	**0.9816**	**0.9827**

In terms of the computation time, the minimum number of iterations required by FWI in the considered scenarios is 24, and it takes approximately 1.5 hours to complete the inversion for one image. This is because the forward modeling is performed in every iteration of the optimization process, which takes about 3.5 minutes. For complicated scenarios such as the two-object cases, more iterations are required, resulting in higher computational costs. However, the well-trained DMRF-UNet is capable of reconstructing one image within 0.01 seconds. Although network training is a computationally expensive process that takes about 16 hours, it only needs to be done once. Using the well-trained network, the permittivity distribution can be reconstructed from the input B-scan in near real-time.

3.3.4 Generalizability and Robustness Tests

1) Zero/Three-Object Scenarios

To evaluate the generalization capability of the proposed DMRF-UNet, 100 three-object and 10 new zero-object scenarios and their corresponding B-scans are produced and directly applied to the well-trained network without additional training. Note that only one-object and two-object scenarios are used in the training phase, and the distributions of the 10 zero-object scenarios are unseen for the network. The SSIM, MSE, MAE, and MRE of the denoised B-scans processed by the first-stage network for signature extraction reach 0.9967, 0.1429, 0.2380, and 1.7041%, respectively. The inversion performance comparison with the existing networks and the SMRF-UNet on the evaluation metrics of these 110 scenarios is provided in Table 3.3. For the MREs, only the 100 three-object scenarios are considered as the $max(|T_{i,j}|)$ (in Equation (3.29)) in the 10 zero-object scenarios are always 0. As shown, the SSIM, MSE, MAE, and MRE of the inversion results using the proposed DMRF-UNet are much better than those of PINet, Enc-Dec, U-Net, and SMRF-UNet.

Table 3.3: Comparison of evaluation metrics of zero/three-object scenarios.

Network	SSIM (↑)	MSE (↓)	MAE (↓)	MRE (%) (↓)
PINet [59]	0.9420	1.8482	0.1216	0.5236
Enc-Dec [61]	0.9609	1.2172	0.0917	0.4138
U-Net [61]	0.9623	1.1701	0.0875	0.3923
SMRF-UNet	0.9649	1.0851	0.0878	0.3548
DMRF-UNet	**0.9691**	**0.9715**	**0.0764**	**0.3140**

The imaging result comparison for the generalized scenarios is presented in Figure 3.9. For the zero-object scenarios, the reconstructed maps using the proposed DMRF-UNet perfectly match the ground truths. However, Enc-Dec, PINet, U-Net, and SMRF-UNet mistakenly identify the noise and clutter as the objects' signatures and reconstruct some fuzzy objects. When three objects are considered, PINet, Enc-Dec, U-Net, and SMRF-UNet can only restore two of the objects or reconstruct a fuzzy map of three objects, especially for the objects with closer permittivity values to the soil. In contrast, the DMRF-UNet is still capable of denoising the B-scan [Figure 3.9(g.1)] and rebuilds the permittivity distribution of three objects with various properties [Figure 3.9(g.2)]. These comparative results demonstrate the better

generalization capability of the DMRF-UNet in diverse subsurface scenarios over the previous networks.

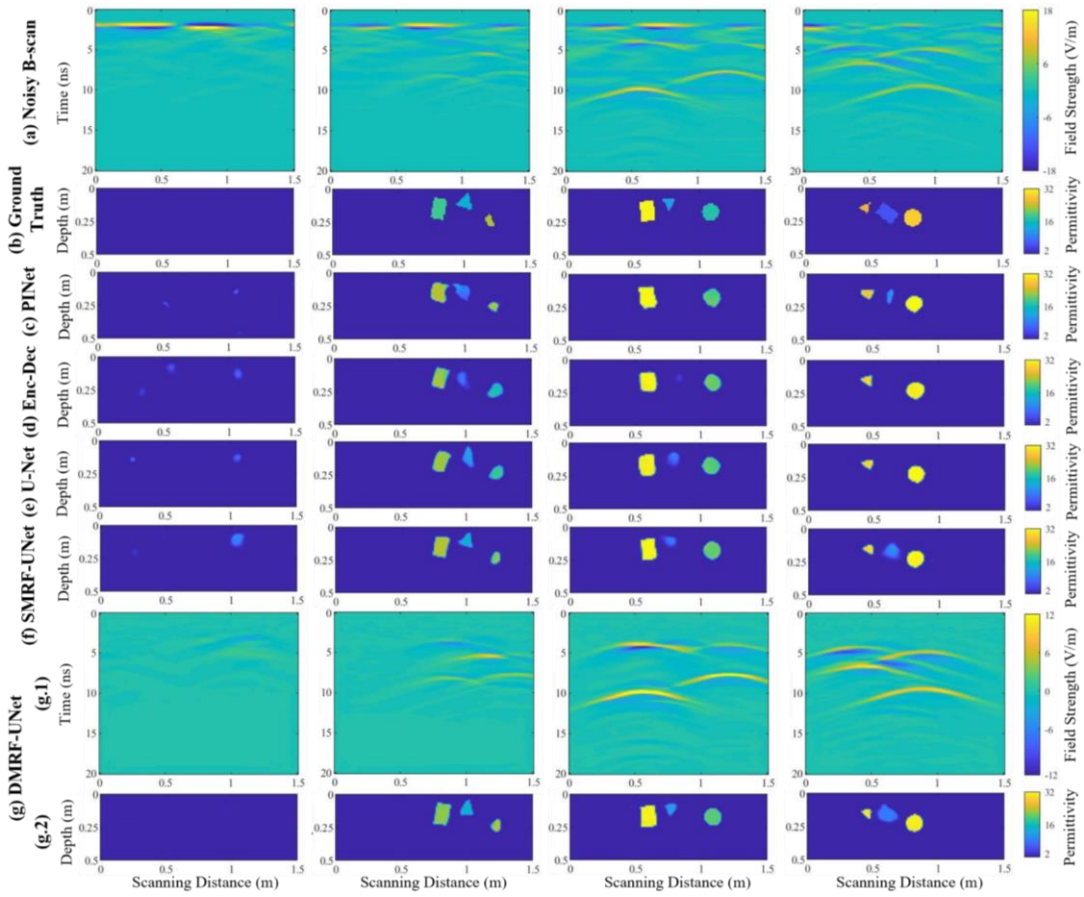


Figure 3.9: Generalizability testing results when no object or three objects are buried. (a) Input noisy B-scans. (b) Ground truths of the permittivity distribution of the subsurface objects. (c) Predicted permittivity distributions from the PINet. (d) Predicted permittivity distributions from the Enc-Dec. (e) Predicted permittivity distributions from the U-Net. (f) Predicted permittivity distributions from the SMRF-UNet. (g) Predicted results from the proposed DMRF-UNet.

2) New Soil Environments

To evaluate the robustness of the proposed method in reconstructing permittivity distributions under new heterogeneous soil conditions, four new Peplinski mixing models are used to test the performance of the DMRF-UNet. Table 3.4 lists the parameters of these soil models, where the network with the soil model described in Section 3.3.1 is listed as “pre-trained”, and the soil models (1)-(4) represent four totally new soil environments. As supervised deep learning-based techniques are data-driven, it is challenging to directly apply the well-trained network to a new dataset vastly different from the training dataset. To adapt the pre-trained network to

new datasets with the soil models (1)-(4), transfer learning is introduced [98]. In particular, a small dataset with the new soil model is generated to fine-tune the pre-trained network. Every dataset includes 360 sets of noisy B-scans, denoised B-scans, and permittivity distributions of subsurface objects. In the fine-tuning process, a small initial learning rate of 0.0001 is used to avoid overfitting. The learning rate is reduced to 99% of its value if the training loss has no drop in the last epoch. After 50-epoch training, the fine-tuned network is used to predict the permittivity distributions from 36 testing B-scans obtained for the new soil models.

Table 3.4: Properties of the utilized heterogeneous soil models and evaluation metrics comparison

Soil Model		(1)	(2)	Pre-Trained	(3)	(4)
Sand Fraction		0.7	0.6	0.5	0.4	0.3
Clay Fraction		0.3	0.4	0.5	0.6	0.7
Water Content Range		0.1%-30%	0.1%-25%	0.1%-20%	0.1%-15%	0.1%-10%
Number of Materials		10	15	20	25	30
Permittivity Range		4.60-13.15	4.08-11.57	3.82-9.99	3.64-8.36	3.57-6.72
Conductivity Range		0.03-0.07	0.02-0.07	0.01-0.07	0.01-0.06	0.01-0.05
SSIM (↑)	#1	0.9961	0.9968	0.9981	0.9968	0.9966
	#2	0.9758	0.9816	0.9845	0.9840	0.9822
MSE (↓)	#1	0.1881	0.1356	0.0674	0.1289	0.1706
	#2	0.6274	0.4152	0.3867	0.3247	0.2996
MAE (↓)	#1	0.3370	0.2449	0.1212	0.2329	0.3003
	#2	0.0552	0.0373	0.0317	0.0328	0.0311
MRE (%) (↓)	#1	2.9823	1.3733	1.3869	1.4743	1.6429
	#2	0.2769	0.2440	0.1642	0.1643	0.1870

As shown in Table 3.4, the SSIMs, MSEs, MAEs, and MREs of the denoising (#1) and inversion (#2) for the four new soil models and the pre-trained model are

compared. For the denoising, the performance of the soil models (2)-(3) is better than the models (1) and (4), as the properties of the soil models (2)-(3) are closer to the one that is well pre-trained using a large diverse set of data. Among these soil models, soil model (1) performs the worst but yet with high SSIM and low MSE, MAE, and MRE. For the inversion, the SSIMs of the soil models (1)-(2) are lower and the MSEs, MAEs, MREs are higher than those of the pre-trained model, but the results are still satisfactory. The performance degradation is due to the increase of the soil water content ratio, resulting in a rise in the reflected signal attenuation, making reconstructing the subsurface permittivity distribution more challenging. For the soil models (3)-(4) with lower signal attenuations, the metrics are quite close to the results of the pre-trained model, the model (4) even slightly outperforms the pre-trained one.

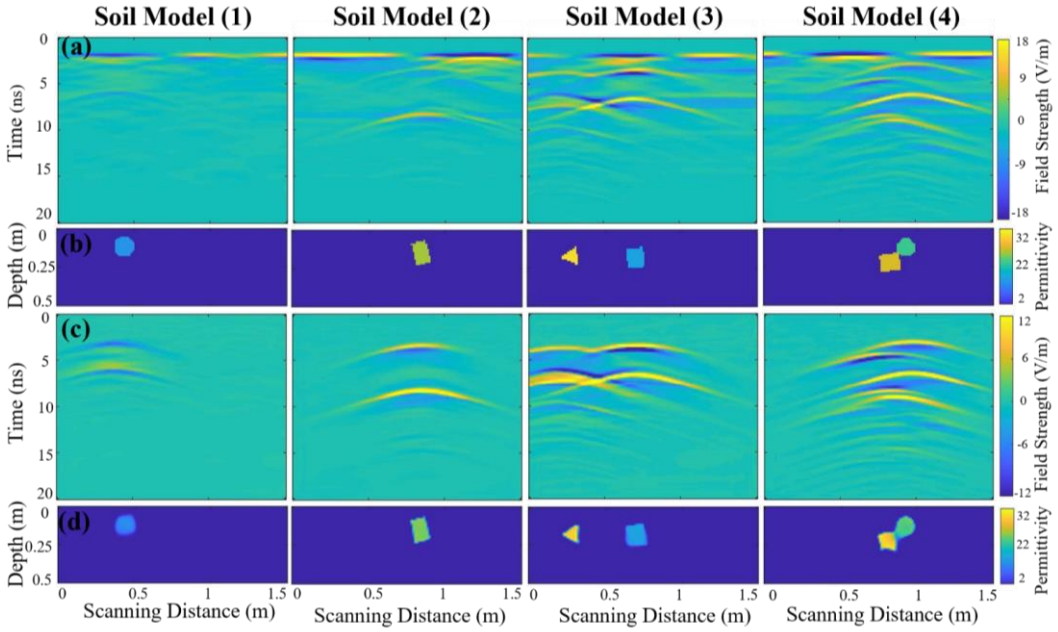


Figure 3.10: Inversion results with four new soil models after fine-tuning the pre-trained model using small sets of new data. (a) Input noisy B-scans. (b) Ground truths. (c) Predicted denoised B-scans. (d) Predicted permittivity distributions.

Figure 3.10 presents the inversion results of the fine-tuned network with the four heterogeneous soil models. Although the clutter and noise in the noisy B-scans [Figure 3.10(a)] are vastly different from those in the pre-trained set, the hyperbolic signatures of objects are still clearly extracted by MRF-UNet1, as shown in Figure 3.10(c). After that, the permittivity distributions are reconstructed by MRF-UNet2. As shown in Figure 3.10(d), the predicted maps with the soil models (1)-(4) match well with their ground truths [Figure 3.10(b)]. These testing results demonstrate that

even when the soil environment is totally new for the network, the permittivity distributions can still be reconstructed with high accuracy via fine-tuning the pre-trained network using an additional small set of new data.

3) New Objects

To further test the generalization capability of the proposed method for new objects via transfer learning, two small datasets for the objects with (1) new permittivity values and (2) new profiles are generated. In the dataset (1), the object's permittivity is randomly selected from [32, 42]. In the dataset (2), to model random shapes of subsurface objects that are very different from those in the pre-training set, the handwritten '1'-shaped objects in the MNIST database [99] are randomly rotated and buried in the soil, as shown in Figure 3.10. Every dataset includes 360 sets of noisy B-scans, denoised B-scans, and permittivity distributions for fine-tuning the pre-trained network described in Section 3.3.1. Other parameter settings are the same as in Section 3.3.4.2). The fine-tuned network predicts the permittivity distributions of the objects with new permittivity values or new profiles from 36 testing B-scans.

Table 3.5: Metrics comparison for objects with new permittivity values and profiles.

Models		SSIM (↑)	MSE (↓)	MAE (↓)	MRE (%) (↓)
Pre-Trained	#1	0.9981	0.0674	0.1212	1.3869
	#2	0.9845	0.3867	0.0317	0.1642
(1) New Permittivity Values	#1	0.9918	0.1211	0.1911	1.0865
	#2	0.9879	1.7613	0.0766	0.2010
(2) New Profiles	#1	0.9883	0.0672	0.1432	1.8047
	#2	0.9827	0.9514	0.0712	0.3319

Table 3.5 lists the evaluation metrics of the two sets of new testing data. Compared to the results of the pre-trained model, the denoising (#1) and inversion (#2) performance for the objects with new permittivity values and new profiles is slightly degraded but satisfactory. Figure 3.11 shows the imaging results for one/two objects with new permittivity values and new profiles. The predicted permittivity distributions shown in Figure 3.11(d) match the corresponding ground truths shown

in Figure 3.11(b). These results demonstrate that when the properties of subsurface objects are out of the pre-training set, the permittivity distributions of the new objects are still accurately reconstructed via the fine-tuned network.

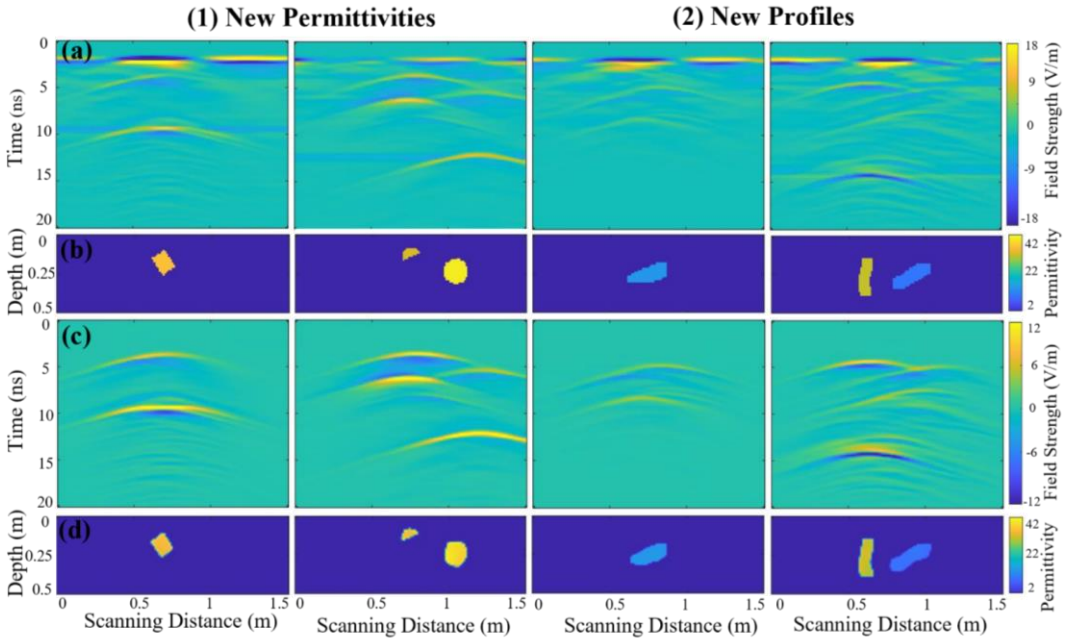
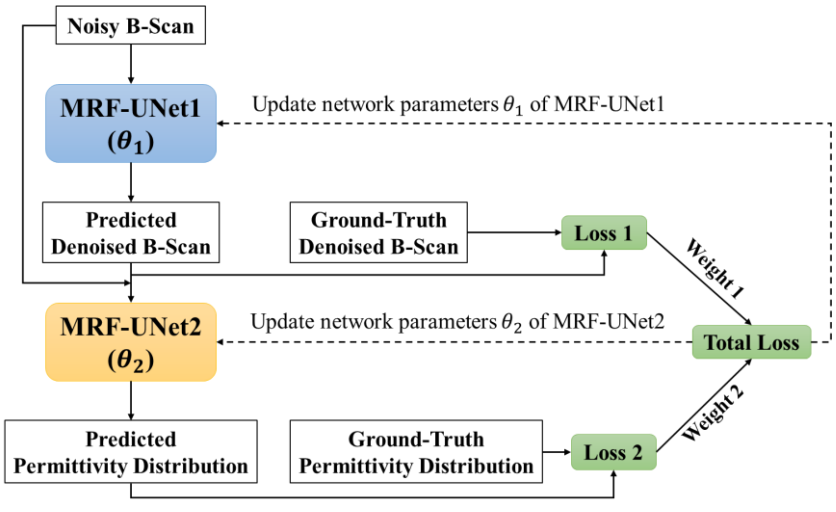


Figure 3.11: Inversion results for objects with (1) new permittivity values and (2) new profiles. (a) Input noisy B-scans. (b) Ground truths. (c) Predicted denoised B-scans. (d) Predicted permittivity distributions.

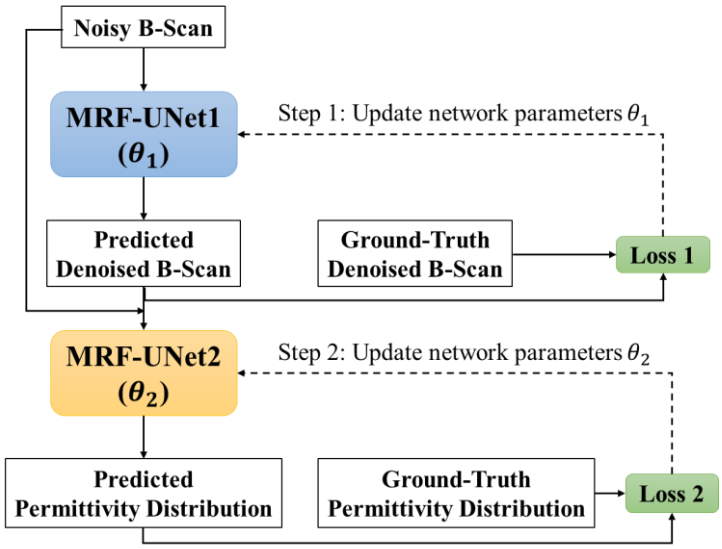
3.3.5 Ablation Study

To prove the effectiveness of the denoising sub-network MRF-UNet1, the end-to-end training method, and the hybrid two-channel input of MRF-UNet2, an ablation study [100] is conducted. We compare the performance of the networks when (i) the MRF-UNet1 for signature extraction is removed, which is equivalent to the SMRF-UNet, (ii) only the denoised B-scans are used to train and test the MRF-UNet2, which means that the input noisy B-scans of the SMRF-UNet in (i) are replaced by the denoised B-scans, (iii) the end-to-end learning that trains the MRF-UNet1 and MRF-UNet2 simultaneously as shown in Figure 3.12(a) is replaced by the separated training that trains the MRF-UNet1 firstly and then trains the MRF-UNet2 as shown in Figure 3.12(b), (iv) the hybrid input of MRF-UNet2 is replaced by the one-channel denoised B-scan, and (v) the final model combines all the proposed components. The comparative metrics are presented in Table 3.6. As shown, the absence of MRF-

UNet1 in (i) leads to a lower SSIM and a higher MSE, MAE, and MRE. This performance degradation is because the noise and clutter in the B-scan under heterogeneous soil conditions interfere with the identification of object signatures. Compared to the inversion performance of using the noisy-B-scan as input in (i), the use of denoised B-scans in (ii) guides the network to extract informative features more effectively, resulting in a decrease in MSE, MAE, and MRE while a slight decrease in SSIM. Still, the inversion performance degrades compared to that of the proposed scheme due to the lack of information compensation. The separate training strategy in (iii) reduces the inversion accuracy compared to the end-to-end training method, which balances the training accuracy of the two sub-networks and compensates for the information loss in predicting the denoised B-scan from MRF-UNet1. The one-channel denoised input in (iv) results in lower inversion accuracy than the hybrid two-channel input, as information loss occurs in the denoising process of MRF-UNet1. The addition of the noisy B-scan in the hybrid two-channel input effectively avoids the information loss in the first-stage denoising process and improves the second-stage inversion performance. Overall, implementing all the components in (v) (the proposed network architecture) leads to the best results with the highest SSIM and the lowest MSE, MAE, and MRE.



(a)



(b)

Figure 3.12: Illustrations of (a) the end-to-end training method and (b) the separate training method for the two-stage network.

Table 3.6: Ablation study on the Inversion Performance.

Study/Metrics	(i)	(ii)	(iii)	(iv)	(v)
Signature Extraction	×	✓	✓	✓	✓
End-to-End Training	✓	×	×	✓	✓
Two-Channel Input	N.A.	×	✓	×	✓
SSIM (↑)	0.9823	0.9811	0.9818	0.9844	**0.9845**
MSE (↓)	0.4252	0.4219	0.4620	0.4008	**0.3867**
MAE (↓)	0.0356	0.0354	0.0381	0.0327	**0.0317**
MRE (%) (↓)	0.1858	0.1826	0.1938	0.1685	**0.1642**

3.4 Tests with Real Measurement Data

3.4.1 Measurement Data Collection and Network Training

The performance of the proposed DMRF-UNet in reconstructing subsurface permittivity distributions is further examined using the field-measured GPR data. As shown in Figures 3.13(a)-(c), the measurement with a commercial GSSI's Utility Scan Pro GPR system is conducted on an outdoor uneven sandy field. Figure 3.13(a) presents the schematic view of the experiment scenario. As illustrated in Figure 3.13(b), the GPR system with a 400-MHz antenna and a control unit is used to collect B-scans in the experiments. Every B-scan obtained along a one-meter scanning trace consists of 88 A-scans, and the number of sampling points in every A-scan is 512. The time window is 20 ns. The buried objects are selected as five wooden objects with various shapes, sizes, and relative permittivity (ε_r) values. The properties of the buried objects are shown in Figure 3.13(c). The ranges of the x-coordinate position, depth, horizontal angle, and vertical angle of the object are selected from [20, 80] cm, [9, 25] cm, [0, 60] degree, and [0, 60] degree, respectively. The permittivity distribution of the field surface is measured by Keysight N1501A Dielectric Probe. As shown in Figure 3.13(d), the relative permittivity along the scanning trace varies from 2.64 to 6.48 due to the difference in humidity.

The measured B-scans are normalized to [0, 1], resized to 128×128, and pre-processed by mean subtraction to form the input noisy B-scans. To obtain the ground truth of the denoised B-scan, a B-scan is measured in an empty area far away from the object region and subtracted from the noisy B-scan. Totally 180 sets of data, including the measured noisy B-scans, the denoised B-scans, and the corresponding permittivity distributions of subsurface objects, are obtained to fine-tune the network pre-trained by simulation data as described in Sections 3.3.1-3.3.2. The initial learning rate is set as 0.0001, and then decreased to 99% of its value to avoid over-fitting if the training loss does not decrease in the last epoch.

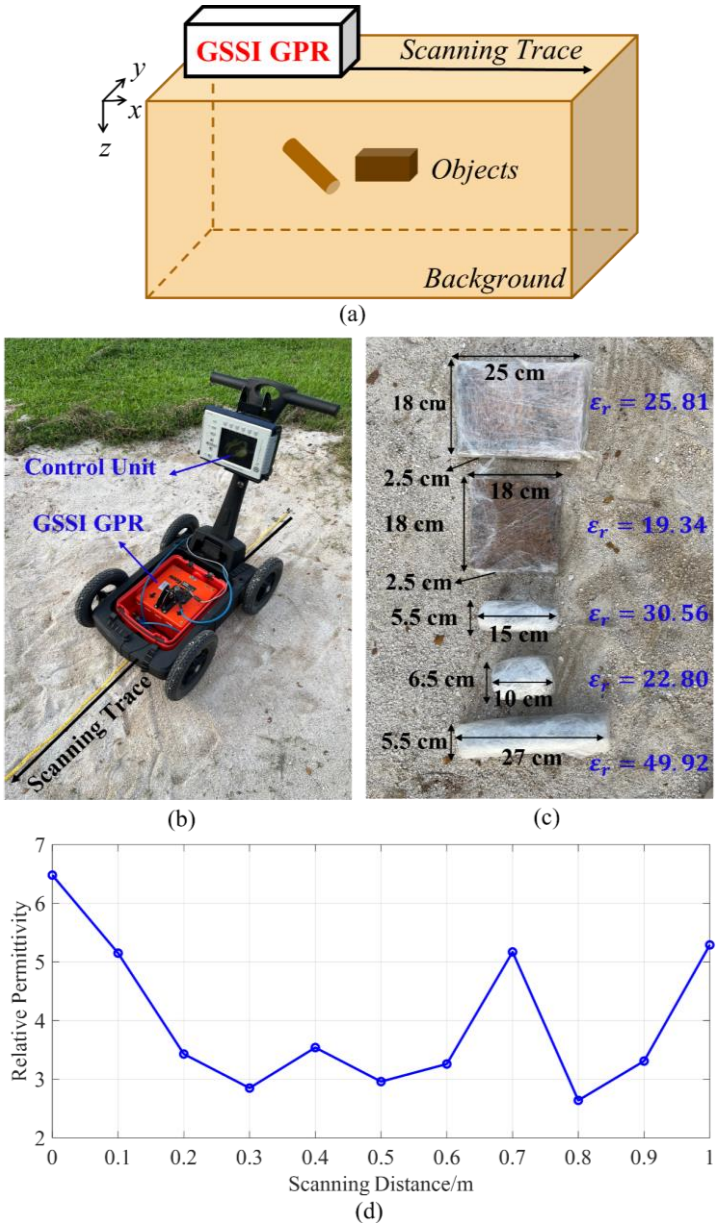


Figure 3.13: The experiment setup for collecting real measurement data. (a) The schematic view of the experiment scenario. (b) The view of the real scenario with the commercial GPR system. (c) Five wooden objects utilized in the experiments. ε_r represents the relative permittivity of the object. (d) The permittivity distribution of the field surface.

3.4.2 Inversion Result Comparison and Analysis

After 200-epoch training, four field-measured noisy B-scans are used to test the fine-tuned network. The SSIM, MSE, MAE, and MRE of the denoising reach 0.9820, 1.3265, 0.7082, 1.0962%, respectively. To quantitatively evaluate the inversion performance of the proposed DMRF-UNet, the comparison with the FWI algorithm, the existing inversion networks and the SMRF-UNet, which are fine-tuned with

transfer learning as well, on evaluation metrics is shown in Table 3.7. For the task of inversion, the MSE (0.3881), MAE (0.0537), and MRE (0.1914%) of DMRF-UNet are much lower than all the results of FWI, PINet, Enc-Dec, U-Net, and SMRF-UNet, while the SSIM (0.9589) is slightly lower than that of PINet. Overall, the performance of the proposed DMRF-UNet on the real measurement data is the best among those of the previous studies.

Table 3.7: Comparison on evaluation metrics of real measurement data.

Method	SSIM (↑)	MSE (↓)	MAE (↓)	MRE (%) (↓)
FWI	0.9412	10.5406	0.3449	0.8371
PINet	**0.9654**	2.4186	0.1503	0.5290
Enc-Dec	0.9478	0.9227	0.0932	0.3456
U-Net	0.9506	0.6564	0.0820	0.3135
SMRF-UNet	0.9579	0.5044	0.0786	0.3037
DMRF-UNet	0.9589	**0.3881**	**0.0537**	**0.1914**

To qualitatively compare the inversion accuracy, the imaging results of the PINet, Enc-Dec, U-Net, FWI, SMRF-UNet, and DMRF-UNet are compared in Figure 3.14. Figures 3.14(a)-(b) present the measured noisy B-scans and the corresponding ground truths of the permittivity distributions of subsurface objects. As shown in Figures 3.14(c)-(g), in Case 1, the PINet, Enc-Dec, U-Net, FWI, and SMRF-UNet roughly restore the size, shape, and position of one circular object but with an inaccurate permittivity value. In Case 2, the PINet, Enc-Dec, U-Net, and SMRF-UNet fail to reconstruct the rectangle object's permittivity distribution, and the FWI cannot accurately restore the object's permittivity and location. In Cases 3-4, the two-separated-object and two-interfaced-object scenarios, the inversion results of PINet, Enc-Dec, U-Net, and FWI deviate dramatically from the ground truths, and the SMRF-UNet cannot precisely restore the shapes and permittivity values especially

for the interfaced objects. In contrast, as shown in Figure 3.14(h.2), the reconstructed permittivity distributions of the proposed DMRF-UNet in Cases 1-4 agree well with the ground truths. Although some objects' boundaries are slightly blurred, the sizes, shapes, positions, and permittivity values of the subsurface objects are restored with high accuracy. This is because the DMRF-UNet effectively extracts the object reflections from the noisy B-scans, as shown in Figure 3.14(h.1), to alleviate the adverse effect of environmental factors on the reconstruction accuracy, while the previous networks without denoising cannot distinguish the objects' reflection signatures and the environmental noise well. These quantitative and qualitative results demonstrate the superior performance of the DMRF-UNet compared to the previous methods on the field-measured data. The good agreement between the predicted and the actual permittivity distributions indicates the effectiveness of the proposed methodology for real-field applications.

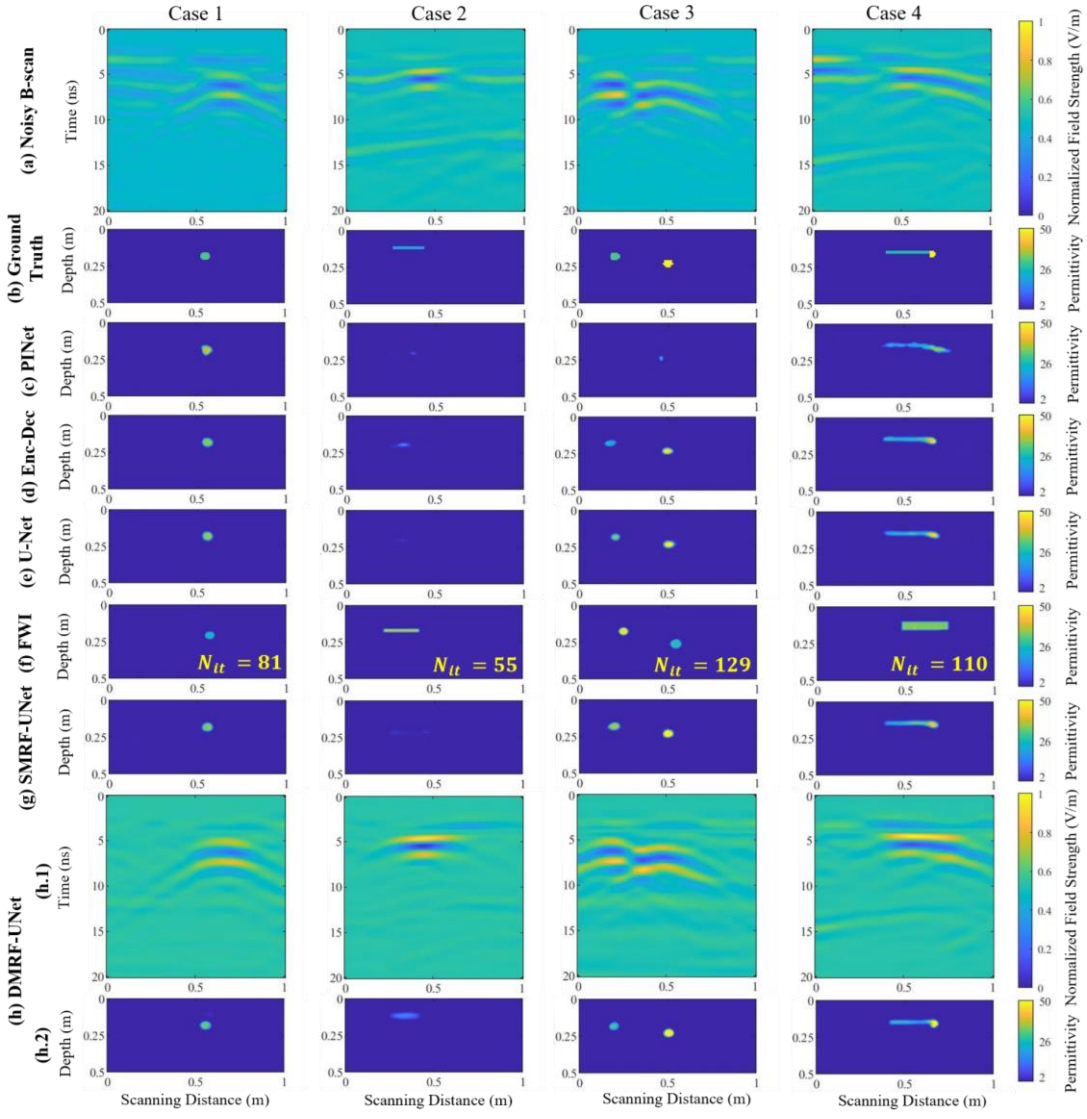


Figure 3.14: Inversion result comparison of real measurement data. (a) Input noisy B-scans. (b) Ground truths of the permittivity distribution of the subsurface objects. (c) Predicted permittivity distributions from PINet. (d) Predicted permittivity distributions from the Enc-Dec. (e) Predicted permittivity distributions from the U-Net. (f) Reconstructed permittivity distributions using the FWI algorithm. The N_{it} stands for the number of iterations required until the convergence of the optimization process. (g) Predicted permittivity distributions from the SMRF-UNet. (h) Predicted results from the proposed DMRF-UNet including (h.1) denoised B-scans from MRF-UNet1, and (h.2) permittivity distributions from MRF-UNet2.

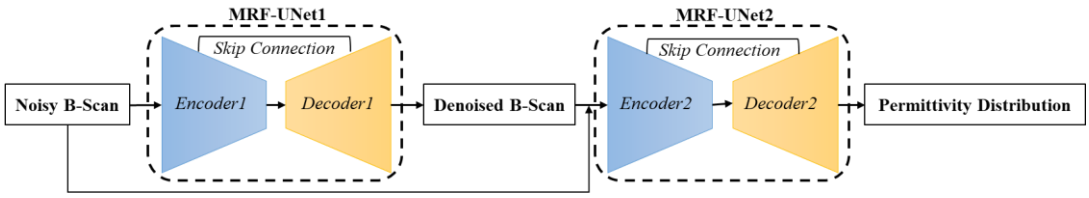
3.5 Discussion and Conclusion

This work proposes a two-stage deep learning-based method for GPR data inversion under heterogeneous soil conditions. The first stage employs a U-shape DNN to extract the object signatures in the noisy B-scan and remove the noise and clutter. The second stage takes the concatenated denoised image and noisy image as input and reconstructs the permittivity distributions of subsurface objects. Convolutions with multiple receptive fields are introduced in both stages to extract multi-scale features from the B-scans. An end-to-end training method is proposed by combining the losses of two stages. The test results demonstrate that the proposed network allows a robust and accurate reconstruction of the permittivity distributions of subsurface objects from the GPR B-scans. The comparison with existing techniques shows the superiority of the proposed scheme, especially for complex scenarios where multiple objects are involved. This work addresses the challenges of GPR inverse problems under heterogeneous soil conditions using deep learning techniques and shows the accuracy and efficiency of the deep learning-based inversion for the measurement data obtained in the real field.

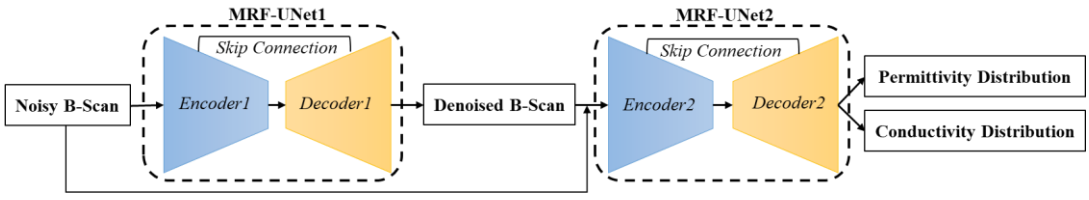
It should be noted that the focus of our GPR data inversion scheme is to reconstruct the permittivity distribution of the subsurface object and the conductivity of the object is ignored. The necessary modifications for the network structure, the loss function, and the training and testing datasets are required when expanding the proposed DMRF-UNet to predict both permittivity and conductivity.

1) The network structure. If both the permittivity and conductivity are considered at the same time, there will be two output images including the permittivity distribution and the conductivity distribution of the inversion network. We would like to provide three potential network structures for simultaneously predicting the

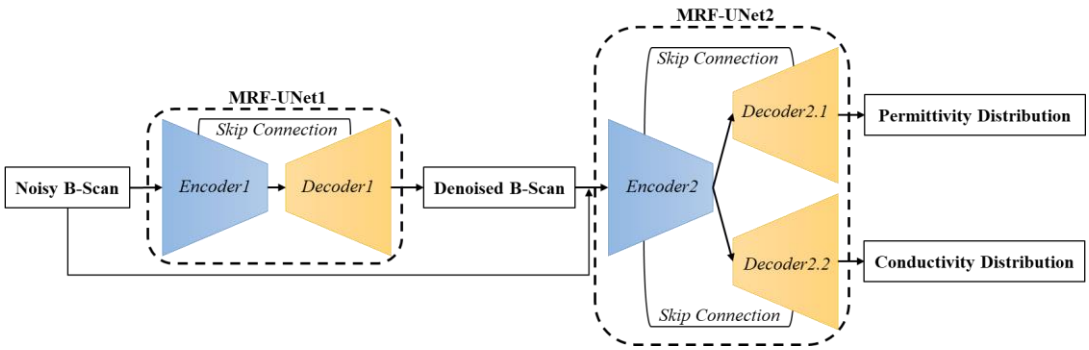
permittivity distribution and the conductivity distribution. Figure 3.15 compares these network structures and shows their difference. As shown in Figure 3.15(a), the proposed DMRF-UNet for reconstructing the permittivity distribution consists of the denoising sub-network MRF-UNet1 and the inversion sub-network MRF-UNet2. Each sub-network includes an encoder and a decoder with skip connections. As shown in Figure 3.15(b), to reconstruct both the permittivity distribution and conductivity distribution, the final one-channel convolutional layer of the proposed DMRF-UNet can be modified as a two-channel convolutional layer to output two images at the same time. As shown in Figure 3.15(c), two decoders including the Encoder2.1 and the Encoder2.2 can be used in the MRF-UNet2 to separately restore the permittivity and conductivity information and output the permittivity and conductivity distribution. As shown in Figure 3.15(d), two inversion networks including the MRF-UNet2.1 and MRF-UNet2.2 can be used to separately find the optimal mappings from the B-scan to the permittivity distribution and from the B-scan to the conductivity distribution.



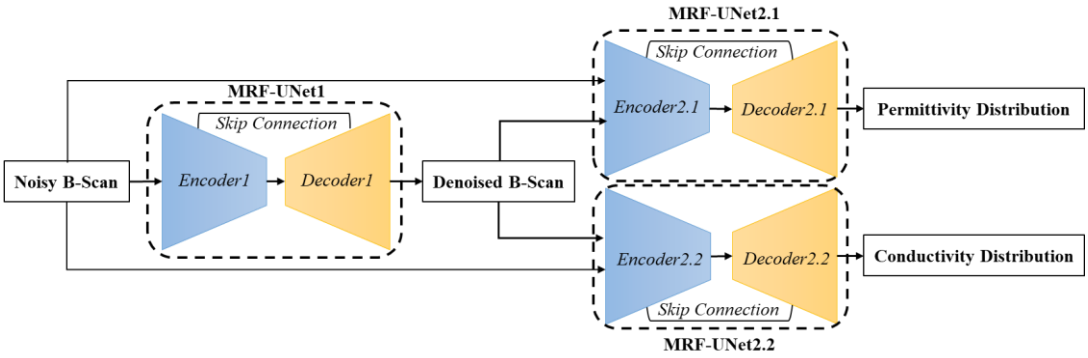
(a)



(b)



(c)



(d)

Figure 3.15. Potential network structures when expanding the proposed network for predicting both the permittivity distribution and the conductivity distribution. (a) Network structure of the proposed DMRF-UNet for predicting the permittivity distribution. (b) Network structure when the final one-channel convolutional layer for predicting the permittivity distribution is replaced by a two-channel channel convolutional layer for predicting the permittivity distribution and the conductivity distribution. (c) Network structure when two decoders are used in the MRF-UNet2 to separately recover the permittivity and conductivity information. (d) Network structure when two inversion networks, the MRF-UNet2.1 and the MRF-UNet2.2, are used to separately predict the permittivity distribution and the conductivity distribution.

2) Loss function for training. For all the provided three network structures, the loss function for training the network can be modified as

$$\ell = \alpha\ell_1 + \beta\ell_2 + \gamma\ell_3 = \alpha\frac{1}{H_1 \times W_1}\sum_{i_1,j_1}\left(Y_{t\,i_1,j_1} - \hat{Y}_{t\,i_1,j_1}\right)^2$$

$$+\beta\frac{1}{H_2 \times W_2}\sum_{i_2,j_2}\left(X_{\varepsilon_r\,i_2,j_2} - \hat{X}_{\varepsilon_r\,i_2,j_2}\right)^2 + \gamma\frac{1}{H_3 \times W_3}\sum_{i_3,j_3}\left(X_{\sigma\,i_3,j_3} - \hat{X}_{\sigma\,i_3,j_3}\right)^2, \quad (3.30)$$

where ℓ is the combined loss. ℓ_1, ℓ_2, and ℓ_3 are the losses of the denoised B-scan, the permittivity distribution, and the conductivity distribution, respectively. α, β, and γ are the weights of ℓ_1, ℓ_2, and ℓ_3, respectively, for balancing their differences in order of magnitude in the training process. Y_t and $\hat{Y}_t$ are the predicted result and the ground truth of the denoised B-scan. X_{ε_r} and $\hat{X}_{\varepsilon_r}$ are the predicted result and the ground truth of the permittivity distribution. X_σ and $\hat{X}_\sigma$ are the predicted result and the ground truth of the conductivity distribution. H_1 and W_1 are the dimensions of the

denoised B-scan, while i_1 and j_1 are the indices. H_2 and W_2 are the dimensions of the permittivity distribution, while i_2 and j_2 are the indices. H_3 and W_3 are the dimensions of the permittivity distribution, while i_3 and j_3 are the indices.

3) Training and testing datasets. For our proposed DMRF-UNet developed for predicting the permittivity distribution, we generated a large set of B-scans and subsurface permittivity distributions to train and test the network. In these scenarios, the conductivities of the subsurface objects are fixed as 0. When we would like to expand the network to predict both the permittivity and conductivity distributions, the conductivities of subsurface objects should be taken into consideration. For the numerical simulation dataset, the conductivities of subsurface objects should be randomly chosen from a given range to ensure the dataset diversity, and then each data set include the conductivity distribution, the permittivity distribution, and the corresponding B-scan. Similarly, for the real measurement dataset used to fine-tune the pre-trained network, the objects' conductivities should be recorded to build the conductivity distribution.

Chapter 4

3D GPR Data Inversion

In this chapter, our second contribution aiming to solve the 3D GPR inverse problem is introduced. A deep learning-based scheme is proposed to first denoise GPR C-scans and then reconstruct subsurface 3D permittivity maps. The introduction, methodology including the 3D network design and learning strategy, the results of the tests on the generalizability and robustness, and the results of the inversion with real measurement data are provided as follows.

4.1 Introduction

In 3D GPR inverse problems, the 3D permittivity map reconstructed from the data obtained by GPR is highly useful to obtain information about the subsurface objects, such as their shapes, sizes, positions, orientations, and permittivity values. Such information would be highly beneficial for subsurface object identification, geophysical defect detection, and health monitoring of the subsurface utilities and living entities. As reviewed in Section 2.1.3, traditional 3D GPR inversion methods, including the back-projection algorithm [37-39], Kirchhoff migration [40], reverse time migration [41-42], and 3D FWI algorithms [43-44], suffer from three aspects: (1) They can only provide an approximation to the objects' positions and shapes but cannot restore their constitutive parameters, or (2) they require high computational costs for iteratively processing 3D GPR data, and (3) only subsurface objects with simple and regular geometries are considered. To improve the accuracy and efficiency of subsurface permittivity map reconstruction, deep leaning-based schemes have been proposed to restore the 2D permittivity map from GPR B-scan images [18, 57-59]. Yet, all these deep learning-based works only focus on the 2D-domain permittivity map restoration, which can only contribute to the sectional information of the subsurface scenarios. 3D-domain information such as the orientation and shape of the object cannot be retrieved, especially when the subsurface object is complex, and one cross-section is not sufficient to generate the

full profile of the object. On the other hand, due to the high computational burden of processing 3D GPR data, it is costly to directly apply the existing complicated 2D DNNs for GPR B-scan processing to 3D cases. Thus, it is imperative to carefully design deep learning models for reconstructing 3D subsurface permittivity maps.

One major challenge for reconstructing subsurface permittivity maps from GPR data is the interference of noise patterns such as direct coupling, reflections from the ground, and environmental noise with object reflections in the GPR data. In the past, many traditional methods have been proposed for GPR data denoising, including mean subtraction [21-22, 101-102], SVD [22-23, 103-105], principal component analysis (PCA) [106-107], and non-negative matrix factorization (NMF) [108-109]. However, the denoising performance of these signal processing algorithms degrades significantly in complicated heterogeneous environments, especially when the field strengths or shapes of noise patterns are similar to those of objects' reflection signatures. To improve the accuracy and efficiency of GPR data denoising, deep learning-based schemes have been proposed [110-117]. In [110-112], DNNs were used to identify regions of interest (RoIs) that contain hyperbolas or to generate a hyperbolic binary mask to highlight the object reflection area. In [113-117], encoder-decoder networks, U-Nets and GANs were employed to predict clutter-free B-scan images that only contain object reflections. Their experimental results demonstrate the effectiveness of deep learning techniques in eliminating environmental noise and preserving object reflections. The comparative studies in [114-115, 117] prove the superiority of deep learning-based methods over the traditional algorithms on GPR data denoising.

In this work, we propose a deep learning scheme, called 3DInvNet, that reconstructs subsurface 3D permittivity maps from GPR C-scans with prior denoising. To the best of our knowledge, this is the first work that performs the 3D subsurface scenario reconstruction from GPR data via a deep learning scheme. The main contributions of the proposed scheme are three-fold. First, a prior 3D denoising network (Denoiser) is designed to suppress noise in GPR C-scans due to heterogeneous soil environments. Different from complicated networks in existing GPR data denoising networks, the Denoiser adopts a small 3D CNN with residual learning and feature attention mechanism to extract subsurface objects' reflection signatures in noisy C-scans. Second, a 3D U-shaped encoder-decoder network (Inverter) is designed to map the

denoised C-scans predicted by the Denoiser into subsurface 3D permittivity maps. Instead of using conventional convolutions with single scale, multi-scale feature aggregation modules are proposed to extract the features of multiple objects with various properties. Third, a three-step separate learning strategy is used to pre-train and fine-tune the Denoiser and Inverter. Test results clearly show that the proposed scheme is capable of denoising C-scans and then reconstructing 3D permittivity maps with high accuracy and efficiency. The test results on new scenarios show good generalizability and robustness of the proposed scheme.

4.2 Methodology

After the GPR system is moved along the parallel traces on the xy plane as demonstrated in Figure 4.1, a 3D C-scan is obtained. As subsurface objects have different geometric and material properties, their corresponding C-scans show different reflection patterns and signatures. If $\tilde{X}$ is the 3D subsurface scenario, $\tilde{H}(\cdot)$ represents forward modeling of the scattering mechanism in the subsurface scenario, and $\tilde{Y}$ is the resultant C-scan, then the forward problem can be described as $\tilde{Y} = \tilde{H}(\tilde{X})$. The inverse problem targets reconstructing subsurface scenarios from C-scans, and is described as

$$\tilde{X} = \tilde{H}^{-1}(\tilde{Y}). \tag{4.1}$$

It converts the EM information stored in the C-scan into a 3D permittivity map to reveal the buried objects' characteristics. The aim of solving the inverse problem based on deep learning is to find an optimal inverse mapping relationship $\tilde{H}^{-1}(\cdot)$ between $\tilde{Y}$ and $\tilde{X}$, which can be described by a 3D DNN model, and then translate $\tilde{Y}$ into $\tilde{X}$ via $\tilde{H}^{-1}(\cdot)$. To overcome the interference of environmental noise, we propose a two-stage scheme, named 3DInvNet, for 3D GPR data inversion. It includes a 3D denoising network (Denoiser) to obtain the noise-free C-scan $\tilde{Y}_d$ from the input noisy C-scan $\tilde{Y}$ and a 3D inversion network (Inverter) to reconstruct $\tilde{X}$ from $\tilde{Y}_d$. The detailed network structures and learning strategy are described as follows.

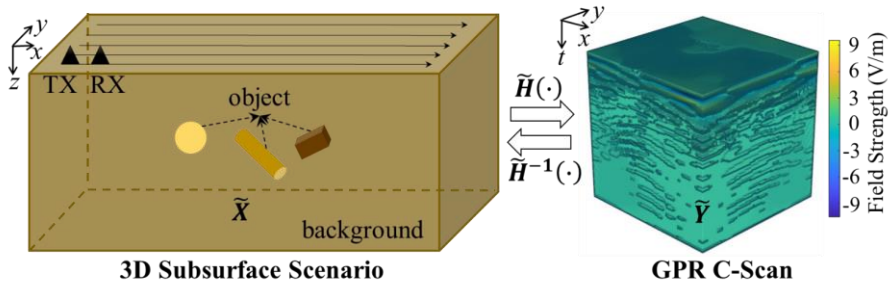


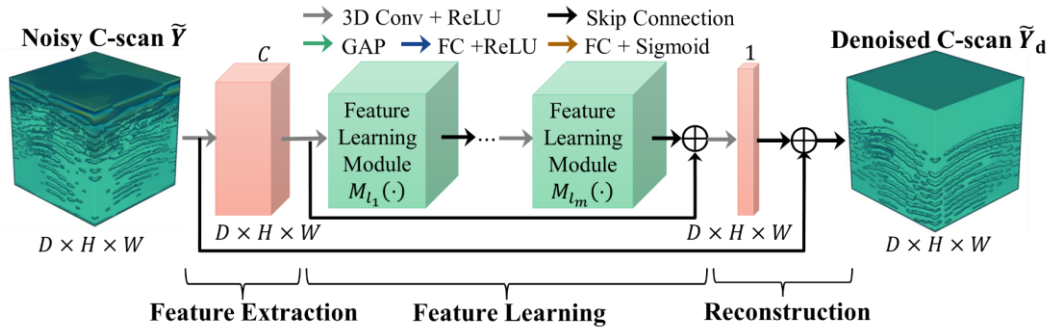
Figure 4.1: Task definition of 3D GPR data inversion. TX and RX denote the transmitter and the receiver in the GPR survey, respectively. $\tilde{X}$ and $\tilde{Y}$ represent the 3D subsurface scenario and the resulting C-scan obtained after the GPR survey performed on the surface, respectively. $\tilde{H}(\cdot)$ and $\tilde{H}^{-1}(\cdot)$. represent the forward and reverse relationship between $\tilde{X}$ and $\tilde{Y}$, respectively.

4.2.1 Denoiser

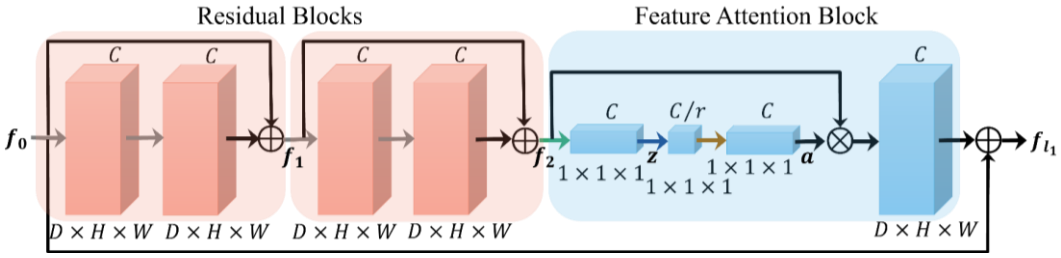
One limitation of applying the 2D DNNs for GPR B-scan denoising in [113-117] to 3D cases is the high computational burden in 3D C-scan image processing. To alleviate this problem, a small 3D CNN with the feature attention mechanism [118], named Denoiser, is employed to denoise the C-scans under heterogeneous soil conditions. As shown in Figure 4.2, the Denoiser consists of three stages: i) initial feature extraction, ii) feature learning, and iii) reconstruction. First, the initial feature extraction module, including one 3×3×3 convolutional layer with C channels and strides of 1×1×1, captures initial features $f_0 \in \mathbb{R}^{C\times D\times H\times W}$ from the input noisy C-scans $\tilde{Y} \in \mathbb{R}^{D\times H\times W}$ via

$$f_0 = \delta\left(\mathcal{F}_{w,b}(\tilde{Y})\right), \tag{4.2}$$

where D, H, and W are spatial dimensions along principal axes. $\mathcal{F}_{w,b}(\cdot)$ represents the 3D convolutional layer with the weight w and bias b. $\delta(\cdot)$ denotes a ReLU activation function to ensure high non-linearity of the network.



(a)



(b)

Figure 4.2: (a) The network structure of Denoiser. (b) The structure of one exampled feature learning module $M_{l_1}(\cdot)$. "Conv", "GAP", and "FC" represent the convolutional layer, global average pooling, and fully connected layer, respectively. C is the channel number of the feature map in the initial feature extraction module. D, H, and W are spatial dimensions along principal axes. f_0 is the input feature map of the feature learning module $M_{l_1}(\cdot)$. f_1 and f_2 are the output feature maps of the first residual block and the second residual block, respectively. z and a represent the channel-wise statistics and the attention vector in the feature attention block, respectively. f_{l_1} is the output feature map of the feature learning module $M_{l_1}(\cdot)$.

Next, m feature learning modules $M_l(\cdot)$ are used to exploit the critical content of the extracted initial features f_0. Every feature learning module comprises two residual blocks and one feature attention block. Each residual block has two 3×3×3 convolutional layers with identity mapping to avoid exploding gradients [119]. The operations of the residual blocks are expressed as

$$f_1 = \delta\left(\mathcal{F}_{w,b}\left(\delta\left(\mathcal{F}_{w,b}(f_0)\right)\right) + f_0\right) \tag{4.3}$$

$$f_2 = \delta\left(\mathcal{F}_{w,b}\left(\delta\left(\mathcal{F}_{w,b}(f_1)\right)\right) + f_1\right). \tag{4.4}$$

After the residual learning, a feature attention block is designed to enhance the

weights of important features in $f_2 \in \mathbb{R}^{C\times D\times H\times W}$. In particular, the global average pooling is first applied to produce channel-wise statistics $z \in \mathbb{R}^{C\times 1\times 1\times 1}$ via

$$z(c) = \frac{1}{D \times H \times W}\sum_{k=1}^{D}\sum_{i=1}^{H}\sum_{j=1}^{W} f_2(c,i,j,k), \tag{4.5}$$

where c is the channel index, and i, j, and k are voxel indices along principal axes. To capture the channel-wise dependencies, a simple gating mechanism [120] that consists of two fully connected layers $\mathcal{L}_w(\cdot)$ with Sigmoid function $\varphi(\cdot)$ is adopted. The obtained attention vector $a \in \mathbb{R}^{C\times 1\times 1\times 1}$ is computed by

$$a = \varphi\left(\mathcal{L}_{w\in\mathbb{R}^{C\times C/r}}\left(\delta\left(\mathcal{L}_{w\in\mathbb{R}^{C/r\times C}}(z)\right)\right)\right), \tag{4.6}$$

where r is a reduction ratio for limiting model complexity. Then the attended feature map is obtained by rescaling $f_2 \in \mathbb{R}^{C\times D\times H\times W}$ with $a \in \mathbb{R}^{C\times 1\times 1\times 1}$. With the residual connection once again, the output feature map $f_{l_1} \in \mathbb{R}^{C\times D\times H\times W}$ of the first feature learning module $M_{l_1}(\cdot)$ is computed by

$$f_{l_1} = a \times f_2 + f_0. \tag{4.7}$$

The final learned features $f_{l_m} \in \mathbb{R}^{C\times D\times H\times W}$ considering m feature learning modules are given by

$$f_{l_m} = M_{l_m}\left(M_{l_{m-1}}\left(\cdots\left(M_{l_1}(f_0)\right)\right)\right). \tag{4.8}$$

Finally, a reconstruction module that includes a one-channel convolutional layer with residual learning is used to reconstruct the denoised C-scan $\tilde{Y}_d \in \mathbb{R}^{D\times H\times W}$ from the learned feature representations via

$$\tilde{Y}_d = \delta\left(\tilde{Y} + \mathcal{F}_{w,b}\left(f_0 + f_{l_m}\right)\right). \tag{4.9}$$

4.2.2 Inverter

After the prior denoising, a 3D inversion network, named Inverter, is designed to reconstruct the 3D subsurface permittivity map from the denoised C-scan $\tilde{Y}_d$. As

shown in Figure 4.3, based on 3D U-Net [121], the structure of the Inverter comprises the encoder and decoder with skip connections. In the conventional structure [121], each encoding or decoding block includes two successive convolutional layers with single scales. To extract features of subsurface objects at different scales, a multi-scale feature aggregation (MSFA) mechanism is proposed in each encoding and decoding block. Every MSFA module has three 3×3×3 convolutional layers with strides of 1×1×1. The increased number of convolutional layers aims to: 1) increase the network's depth to enhance its representation capability for the non-linear mapping from C-scans to 3D permittivity maps and 2) extract larger-scale features of objects' reflections. The receptive field size r_η of the output feature map f_{r_η} of the ηth convolutional layer in the MSFA module is computed by Equation 3.24. k and s are fixed as 3 and 1, respectively. Then different receptive field sizes result in multiple scales. As shown Figure 4.3, we concatenate $f_{r_1} \in \mathbb{R}^{\tilde{C}\times D\times H\times W}$ with the receptive field size of $r_1 \times r_1 \times r_1$, $f_{r_2} \in \mathbb{R}^{\tilde{C}\times D\times H\times W}$ with the receptive field size of $r_2 \times r_2 \times r_2$, and $f_{r_3} \in \mathbb{R}^{\tilde{C}\times D\times H\times W}$ with the receptive field size of $r_3 \times r_3 \times r_3$ along the channel dimension in each encoding and decoding block. The fused multi-scale feature map $f_{ms} \in \mathbb{R}^{3\tilde{C}\times D\times H\times W}$ is expressed as

$$f_{ms} = Concat\left(f_{r_1}, f_{r_2}, f_{r_3}\right). \tag{4.10}$$

Unlike the introduction of a large amount of additional convolutional layers in parallel that involves highly increased computational cost in the proposed 2D inversion network as described in Section 3.2, the MSFA module directly combines the feature maps from successive convolutional layers with different receptive fields, which efficiently captures multi-scale features of the reflection patterns in GPR C-scans due to various properties of subsurface objects.

As shown in Figure 4.3, the encoder consists of n encoding blocks. Each encoding block is composed of one MSFA module and a 2×2×2 max pooling layer with strides of 2×2×2. The MSFA module extracts the objects' features from input C-scans and the compressed representation of those features is obtained after down-sampling via the pooling layer. The channels of n encoding blocks are set as $[\tilde{C}, 2\tilde{C}, \cdots, 2^{n-1}\tilde{C}]$. Each convolutional layer is followed by batch normalization (BN) [122] and ReLU activation. In the BN, each batch is standardized with its mean and standard deviation to accelerate the convergence. After the encoder, one MSFA module with $2^n\tilde{C}$

channels is used to bridge the encoder and decoder.

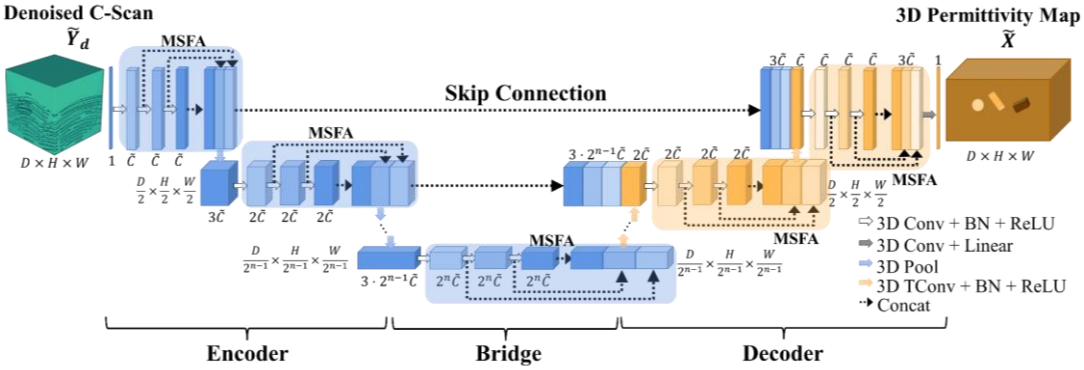


Figure 4.3: The network structure of Inverter. “Conv”, “BN”, “ReLU”, “Linear”, “Pool”, “TConv”, “Concat”, and “MSFA” represent the convolutional layer, batch normalization, rectified linear unit activation, linear activation, max-pooling layer, transposed convolutional layer, concatenation operation, and multi-scale feature aggregation module, respectively. $\tilde{C}$ is the channel number of the feature map in the first MSFA module. D, H, and W are spatial dimensions along principal axes.

The symmetrical decoder includes n decoding blocks. Each decoding block includes a 2×2×2 transposed convolutional layer with strides of 2×2×2, carried out for up-sampling, and a MSFA module, used to reconstruct the permittivity maps from the compressed representations of objects’ signatures. All the transposed convolutional layers and the convolutional layers in the MSFA module are followed by BN and ReLU activations. The channels are symmetrically set as $[2^{n-1}\tilde{C}, 2^{n-2}\tilde{C}, \cdots, \tilde{C}]$. Skip connections are performed by concatenating all the convolutional layers with equal resolution in the encoder, which can provide high-resolution features and compensate for the information loss in the down-sampling process in the encoder. In the final stage, a 3×3×3 one-channel convolutional layer with strides of 1×1×1 followed by a linear activation outputs the desired 3D permittivity map $\tilde{X}$.

4.2.3 Learning Strategy

To save the computational costs of training the 3D networks, we adopt a separate learning strategy for training the Denoiser firstly and then training the Inverter. The details are described as follows.

Step 1: Pre-train Denoiser. A large diverse set of noisy and noise-free C-scans are used to pre-train the Denoiser. The MSE between the predicted denoised C-scan $\tilde{Y}_d$

and the ground truth $\hat{\tilde{Y}}_d$ is used as the loss function:

$$\tilde{\ell}_1\left(\tilde{Y}_d, \hat{\tilde{Y}}_d\right) = \frac{1}{D \times H \times W} \sum_{k=1}^{D} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(\tilde{Y}_{d_{i,j,k}} - \hat{\tilde{Y}}_{d_{i,j,k}}\right)^2. \tag{4.11}$$

The Adam optimizer is employed to update the network parameters for minimizing the denoising loss $\tilde{\ell}_1$.

Step 2: Pre-train Inverter. The ground truth of noise-free C-scan $\hat{\tilde{Y}}_d$ is used as input data of the Inverter in the pre-training phase. With the guidance of noise-free C-scans, the Inverter can accurately capture object's reflection signatures while overcoming the interference of noise patterns due to reflections from the heterogeneous soil. The MAE between the predicted permittivity map $\tilde{X}$ and the ground truth $\hat{\tilde{X}}$ is used as the loss function:

$$\tilde{\ell}_2\left(X, \hat{X}\right) = \frac{1}{D \times H \times W} \sum_{k=1}^{D} \sum_{i=1}^{H} \sum_{j=1}^{W} \left|\tilde{X}_{i,j,k} - \hat{\tilde{X}}_{i,j,k}\right|. \tag{4.12}$$

The training of Inverter minimizes the inversion loss $\tilde{\ell}_2$ via Adam optimizer. Then in the testing phase, the denoised C-scan $\tilde{Y}_d$ obtained from the Denoiser is fed into Inverter and the corresponding 3D permittivity map $\tilde{X}$ is reconstructed.

Step 3: Fine-tune the pre-trained networks. To enhance the networks' generalizability for new scenarios that are vastly different from the pre-training dataset, transfer learning [98] is further employed via fine-tuning the pre-trained networks obtained in steps 1-2. First, a small additional set of data with new scenarios are generated. Second, the pre-trained networks are set as the initial states and the network parameters are further updated based on the loss functions (4.11) and (4.12) using the new training dataset until convergence. Finally, the fine-tuned networks with the transferred data domain can predict the denoised C-scans and reconstruct 3D permittivity maps for the new scenarios.

4.3 Numerical Experiments

4.3.1 Dataset Generation

Dataset I. To train and test the proposed networks, a dataset covering diverse subsurface scenarios is generated. Each data set includes a noisy C-scan $\tilde{Y}$, a noise-free C-scan $\hat{Y}_d$, and a corresponding 3D subsurface permittivity map $\hat{X}$. The numerical dataset is generated by an open-source 3D FDTD simulator, gprMax [65-66]. As shown in Figure 4.1, a 1×1×0.26 m^3 soil environment is considered. The resolution is 2.5×2.5×2.5 mm^3. The relative permittivity and conductivity of the soil are set as 4 and 0, respectively. A Hertzian dipole transmits a Ricker signal with a center frequency of 1 GHz. The time window is set to 15 ns. A probe positioned 10 cm away from the dipole is used to collect the reflected signals. Both the dipole and probe are 2 cm above the ground. The scanning trajectories are shown as the black lines in Figure 4.1. There are 12 scanning lines with ten evenly spaced scanning points on every line, resulting in 120 scanning points in total. The shapes of subsurface objects are randomly selected as cylinder, sphere, or box. The permittivity of each object is selected in the range of [8, 27]. For the cylinder, the radius and length are randomly selected from the intervals of [0.02, 0.05] m ([0.29, 0.73] wavelength) and [0.01, 0.33] m ([0.15, 4.84] wavelength), respectively. For the sphere, the radius is randomly selected from [0.02, 0.05] m ([0.29, 0.73] wavelength). For the box, all the edge lengths are randomly selected from [0.04, 0.1] m ([0.59, 1.47] wavelength). It should be noted that the wavelength is calculated based on the highest significant frequency of 2.2 GHz. The subsurface object is allowed to be positioned in a domain of 0.4×0.4×0.26 m^3 centered in the middle of the soil to guarantee complete hyperbolic patterns. The simulated C-scans are processed by time-zero correction and mean subtraction to remove direct coupling signals and reflections from the ground, and then used as ground-truth noise-free C-scans $\hat{\hat{Y}}_d$.

To model environmental noise due to the reflections from the heterogeneous soil, 3D Peplinski mixing models [70-72] are used to build soil environments without any objects. The soil properties are set as sand fraction 0.6, clay fraction 0.4, bulk density 2 g/cm^3, and sand particle density of 2.66 g/cm^3. 50 different materials described by 50 Debye functions over a range of water volumetric fractions from 0.1% to 10% are randomly distributed in the soil, producing a relative permittivity range of [3.63, 7.42] and a conductivity range of [0.01, 0.05] S/m. To increase the diversity of noise patterns, ten random heterogeneous distributions are generated, and the simulated C-scans contain only environmental noise. After time-zero correction and mean

subtraction, the noise-only C-scan is added to the noise-free C-scan $\hat{\tilde{Y}}_d$ to form a synthetic noisy C-scan $\tilde{Y}$. Finally, 5850 sets of [$\tilde{Y}$, $\hat{\tilde{Y}}_d$, $\hat{\tilde{X}}$] (1950 one-object scenarios and 3900 two-object scenarios) are produced for pre-training and 150 (50 one-object scenarios and 100 two-object scenarios) are generated for testing.

Dataset II. To further apply the proposed method to heterogeneous scenarios, rather than the direct addition in dataset I, the objects are buried in the heterogeneous soil and the GPR scanning on the xy plane is conducted to obtain the noisy C-scan $\tilde{Y}$. It should be noted that the random distribution of the heterogeneous soil is new as well. The ground-truth noise-free C-scan $\hat{\tilde{Y}}_d$ is obtained via subtracting the noise-only C-scan from the noisy C-scan. The background permittivity in the ground-truth permittivity maps is set to 5.72, which is the average relative permittivity of the 50 materials in the heterogeneous soil model. With randomly selected properties of subsurface objects, 285 sets of [$\tilde{Y}$, $\hat{\tilde{Y}}_d$, $\hat{\tilde{X}}$] (95 one-object scenarios and 190 two-object scenarios) are generated to fine-tune the pre-trained Denoiser and Inverter, and 15 (5 one-object scenarios and 10 two-object scenarios) are generated for testing.

4.3.2 Implementation Details

The proposed scheme is implemented on PyTorch [123] and executed on an NVIDIA GeForce RTX 3090 GPU. Every C-scan is normalized to the range of [0, 1]. Both the C-scans and permittivity maps are resized to 128×128×128. The noisy C-scan $\tilde{\tilde{Y}}$ and noise-free C-scans $\hat{\tilde{Y}}_d$ in dataset I are used for pre-training Denoiser based on Equation (4.11). The noise-free C-scans $\hat{\tilde{Y}}_d$ and 3D permittivity maps $\hat{\tilde{X}}$ in dataset I are employed for pre-training Inverter based on Equation (4.12). m and C in the Denoiser are set as 2 and 8, respectively. n and $\tilde{C}$ in the Inverter are set as 4 and 8, respectively. The learning rate is initially 0.001 and decreased to 98% of its value if the training loss has no drop in the last epoch. The Denoiser and Inverter are separately pre-trained for 100 epochs. The models with the lowest validation losses are used to test the performance. In the testing stage, the noisy C-scan is fed into the pre-trained Denoiser, then the denoised C-scan $\tilde{Y}_d$ is input into the pre-trained Inverter, which finally outputs the 3D permittivity map $\tilde{X}$. Using the pre-trained Denoiser and Inverter as the initial state, the networks are further fine-tuned with dataset II. To avoid over-fitting, the learning rate is set as small as 0.0006 and

decreased to 99% of its value if the training loss has no drop in the last epoch.

The 3D SSIM, peak signal-to-noise ratio (PSNR), MAE, and RE are defined to analyze the denoising performance of Denoiser. The 3D SSIM, MSE, MAE, and mean absolute percentage error (MAPE) [124] are used to evaluate the inversion performance of Inverter. These metrics are expressed as:

$$SSIM(\tilde{T},\tilde{P}) = \frac{(2\mu_{\tilde{P}}\mu_{\tilde{T}} + c_1)(2\sigma_{\tilde{P}\tilde{T}} + c_2)}{(\mu_{\tilde{P}}^2 + \mu_{\tilde{T}}^2 + c_1)(\sigma_{\tilde{P}}^2 + \sigma_{\tilde{T}}^2 + c_2)}, \tag{4.13}$$

$$MSE(\tilde{T},\tilde{P}) = \frac{1}{D \cdot H \cdot W}\sum_{k=1}^{D}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(\tilde{P}_{i,j,k} - \tilde{T}_{i,j,k}\right)^2, \tag{4.14}$$

$$PSNR = 10\log_{10}\frac{1}{MSE}, \tag{4.15}$$

$$MAE(\tilde{T},\tilde{P}) = \frac{1}{D\cdot H\cdot W}\sum_{k=1}^{D}\sum_{i=1}^{H}\sum_{j=1}^{W}\left|\tilde{P}_{i,j,k} - \tilde{T}_{i,j,k}\right|, \tag{4.16}$$

$$RE(\tilde{T},\tilde{P}) = \frac{\sqrt{\sum_{k=1}^{D}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(\tilde{P}_{i,j,k} - \tilde{T}_{i,j,k}\right)^2}}{\sqrt{\sum_{k=1}^{D}\sum_{i=1}^{H}\sum_{j=1}^{W}\tilde{T}_{i,j,k}^2}} \cdot 100\%, \tag{4.17}$$

$$MAPE(\tilde{T},\tilde{P}) = \frac{1}{D\cdot H\cdot W}\sum_{k=1}^{D}\sum_{i=1}^{H}\sum_{j=1}^{W}\frac{\left|\tilde{P}_{i,j,k} - \tilde{T}_{i,j,k}\right|}{\left|\tilde{T}_{i,j,k}\right|} \cdot 100\%. \tag{4.18}$$

In these equations, $\tilde{P}$ represents the predicted result, which can be the denoised C-scan $\tilde{Y}_d$ obtained from the Denoiser or the reconstructed 3D permittivity map $\tilde{X}$ obtained from the Inverter. $\tilde{T}$ represents the ground truth $\hat{\tilde{Y}}_d$ or $\hat{\tilde{X}}$. $\mu_{\tilde{P}}$ and $\mu_{\tilde{T}}$ are means of $\tilde{P}$ and $\tilde{T}$, respectively. $\sigma_{\tilde{P}}$ and $\sigma_{\tilde{T}}$ are variances of $\tilde{P}$ and $\tilde{T}$, respectively. $\sigma_{\tilde{P}\tilde{T}}$ is the covariance of $\tilde{P}$ and $\tilde{T}$. Smaller MSE, MAE, RE, and MAPE, and larger SSIM and PSNR indicate higher accuracy.

4.3.3 Result Analysis

Quantitative results. The average evaluation metrics of the testing samples for 3D denoising are listed in Table 4.1. For the testing data with additional noise in dataset I, the well-trained Denoiser achieves high SSIM and PSNR and low MAE and RE. For the noisy simulated data in dataset II, using the fine-tuned network, the denoising

accuracy is still high. Even though the reflection patterns in C-scans of dataset II is different from those in dataset I, the transfer learning can adapt the network pre-trained by dataset I to the new dataset II, which effectively enhances the generalizability of the network.

Table 4.1: Denoising evaluation metrics on datasets I and II.

Dataset	# of Testing Samples	SSIM ($\times 10^{-2}$) (↑)	PSNR (dB) (↑)	MAE ($\times 10^{-4}$) (↓)	RE (%) (↓)
I	150	99.96	62.99	3.78	0.14
II	15	99.98	66.95	2.08	0.09

Table 4.2: Inversion evaluation metrics on datasets I and II.

Groups	Amount		SSIM ($\times 10^{-2}$) (↑)		MSE ($\times 10^{-2}$) (↓)		MAE ($\times 10^{-2}$) (↓)		MAPE (%) (↓)	
	I	II	I	II	I	II	I	II	I	II
i	50	5	99.91	99.65	3.04	1.81	0.49	0.76	0.07	0.11
ii	67	6	99.73	99.26	8.90	8.06	1.10	1.64	0.15	0.20
iii	33	4	99.63	99.10	15.12	19.30	1.74	2.89	0.21	0.31
i, ii, and iii	150	15	99.77	99.34	8.31	8.98	1.04	1.68	0.14	0.20

Table 4.3: Computation time of the proposed 3DInvNet.

Network	# of Parameters ($\times 2^{20}$)	Pre-Training Time (h)	Fine-Tuning Time (h)	Testing Time (s)
Denoiser	0.01	27.50	2.05	0.20
Inverter	3.06	58.33	3.70	0.39

To quantitatively analyze the accuracy of 3D permittivity map reconstruction using

the Inverter, we classify the testing data into three groups: i) one-object cases, ii) two-separated-object cases, and iii) two-overlapping-object cases. The metrics on the three types of data in datasets I and II are shown in Table 4.2. Clearly, for both datasets I and II, the SSIMs, MSEs, MAEs, and MAPEs of one-object cases (group i) are the best. For the two-separated-object cases (group ii), the SSIMs are lower and MSEs, MAEs, and MAPEs are higher than those in one-object cases. The performance degradation is because the subsurface scenarios with two objects are more complex, which makes reconstructing two-object permittivity maps much more challenging than one-object cases. The hyperbolic signatures in the C-scan for two overlapping-object cases (group iii) are even more complicated, increasing the difficulty of GPR data inversion. This is evidenced by the decreases in SSIMs and increases in MSEs, MAEs, and MAPEs. On average, the high SSIM and low MSE, MAE, and MAPE of the total 150 testing data combining all these three cases in dataset I prove the capability of Inverter in accurately reconstructing permittivity maps. Compared to the results of dataset I, the average SSIM of the 15 testing sets in dataset II is slightly lower and the average MSE, MAE, and MAPE are higher but remain satisfactory.

As for the inversion efficiency, the proposed scheme outperforms the classical iterative algorithms. For example, in the most conservative scenario, one iteration of FWI requires computing a C-scan for a given permittivity map, which takes over 40 minutes when using the simulator gprMax on a GPU platform in our case. Moreover, FWI often requires more than one iteration [43-44], which is certainly time-consuming. However, as shown in Table III, the total testing time using the proposed 3DInvNet, including the Denoiser and Inverter, to generate one permittivity map is only 0.59 s, which is at least 4000 times less than that required by one iteration of FWI. Although the pre-training and fine-tuning are computationally expensive, they only need to be done once. Using the well-trained networks, the 3D permittivity maps can be predicted from the C-scans in near real-time.

Imaging results. For dataset I, the imaging results of the denoised C-scans obtained from Denoiser are shown in Figure 4.4. Cases 1-2, Cases 3-4, and Cases 5-6 present the one-object scenarios (group i), two-separated-object scenarios (group ii), and two-overlapping-object scenarios (group iii), respectively. To visualize the denoised C-scans clearly, selected B-scans [(5)-(8)] that include the main reflection signatures in

each C-scan are plotted. Compared with the noisy C-scans, the noise due to reflections from the heterogeneous soil environments is highly suppressed in the denoised C-scans. The reflection patterns of subsurface objects are extracted and used as the inputs of Inverter to reconstruct 3D permittivity maps.

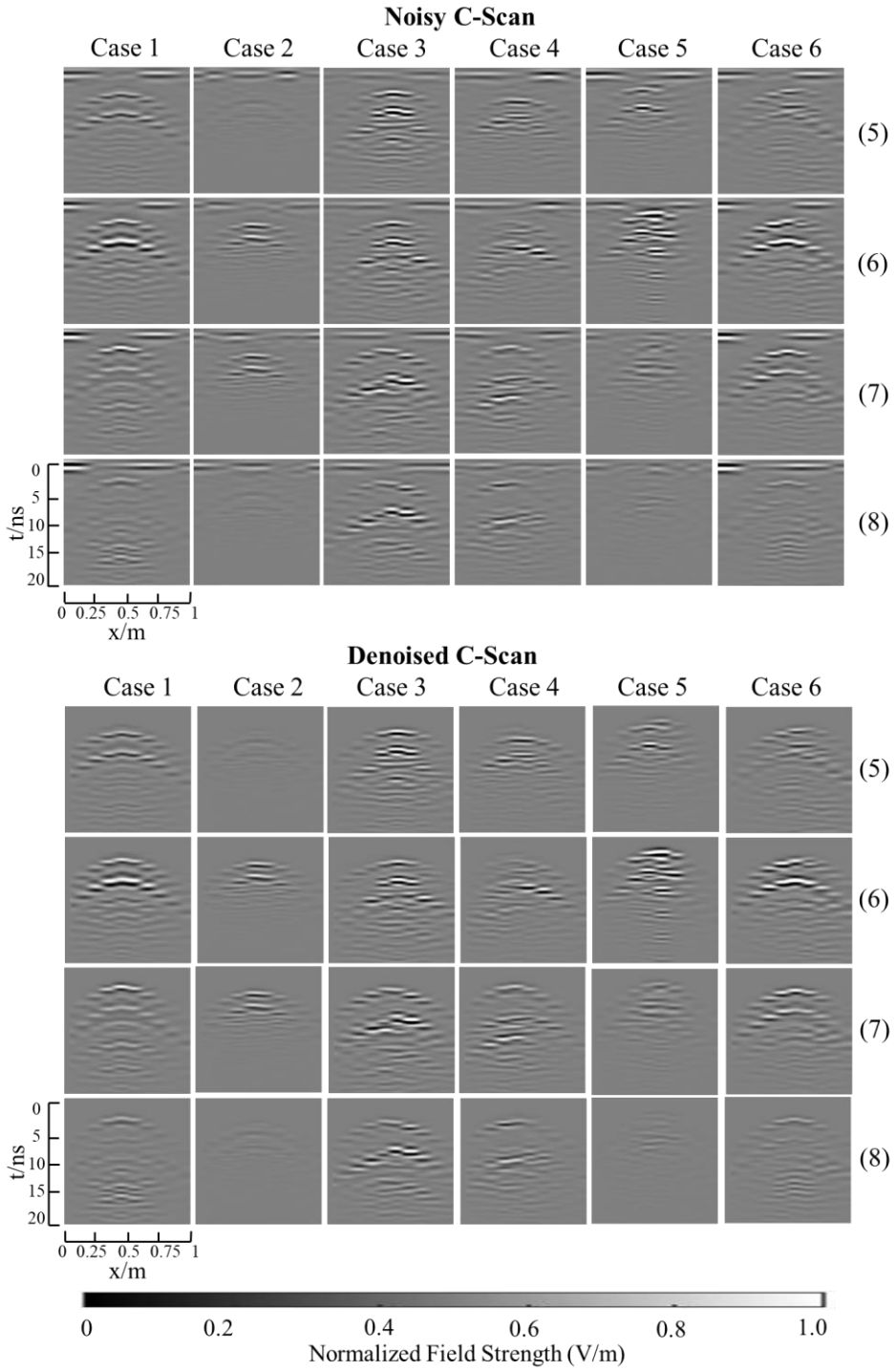


Figure 4.4: The imaging results of denoised C-scans predicted by Denoiser with dataset I. It should be noted that each C-scan includes 12 B-scans and selected B-scans (5)-(8) with the main reflection

signatures are shown. Cases 1-2, Cases 3-4, and Cases 5-6 present the one-object scenarios, two-separated-object scenarios, and two-overlapping-object scenarios, respectively.

As shown in Figures 4.5-4.6, the 3D permittivity maps predicted by the Inverter [Figure 4.5(e) and Figure 4.6(e)] are compared with the ground-truth 3D permittivity maps [Figure 4.5(a) and Figure 4.6(a)]. Both the 3D geometry and one-slice of the permittivity map are presented. In the 3D geometry, the soil environment is set as transparent, and the objects are shown in brown. Cases 1-2 show the results for one-object scenarios. The predicted geometries match well with the actual geometries in the ground truths. To further evaluate the accuracy in estimating the permittivity map, the ground truth and prediction of permittivity maps on a cross-section are compared. It is observed that the predicted permittivity map, including the value and position, is quite close to the ground truth. The imaging results of the two-separated-object cases are shown as Cases 3-4. Compared to the one-object scenarios, two objects introduce more complicated signatures in the C-scan, such as the interleaved hyperbolas and misleading multiple reflected noise. As presented, the proposed method still achieves high accuracy. Even though these objects buried in the soil have various shapes, locations, sizes, and permittivity values, both the predicted 3D geometry and one-slice permittivity map match well with the ground truths.

To verify the high performance in the resolution of our proposed scheme, the inversion results for overlapped and inhomogeneous objects are shown as Cases 5-6 in Figure 4.6. Due to more complex subsurface distributions, reconstructing 3D permittivity maps from the C-scans becomes more challenging. The imaging results show that the predicted permittivity maps as shown in Figure 4.6(e) are close to the ground truth ones as shown in Figure 4.6(a). Even though there is some mismatch on the objects' edges, the subsurface objects can still be clearly recognized.

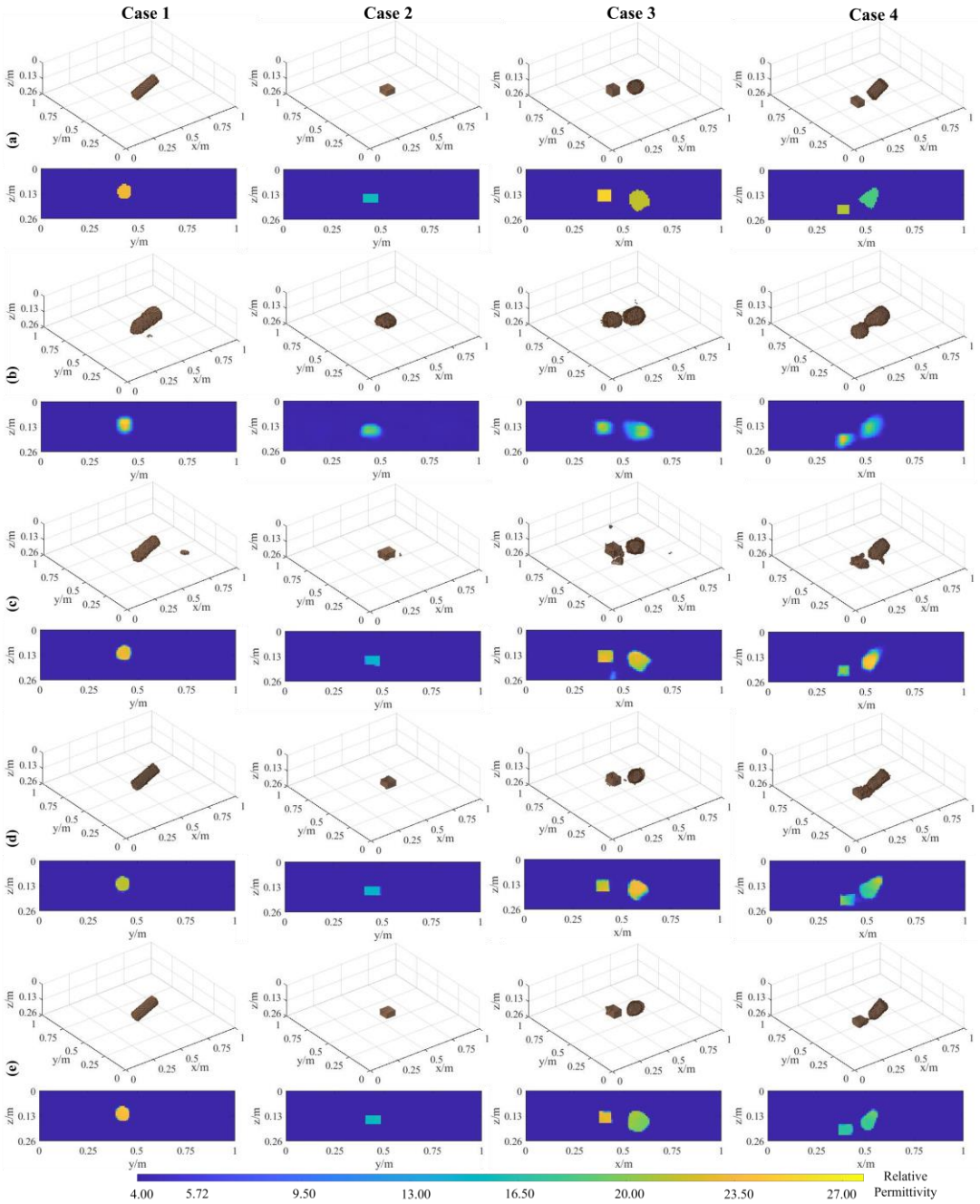


Figure 4.5: The imaging results of 3D permittivity maps predicted by the proposed 3DInvNet for one-object scenarios (Cases 1-2) and two-separated-object scenarios (Cases 3-4) in dataset I. While (a) shows the ground truths, (b)-(e) show the predictions using Models (1)-(4), respectively.

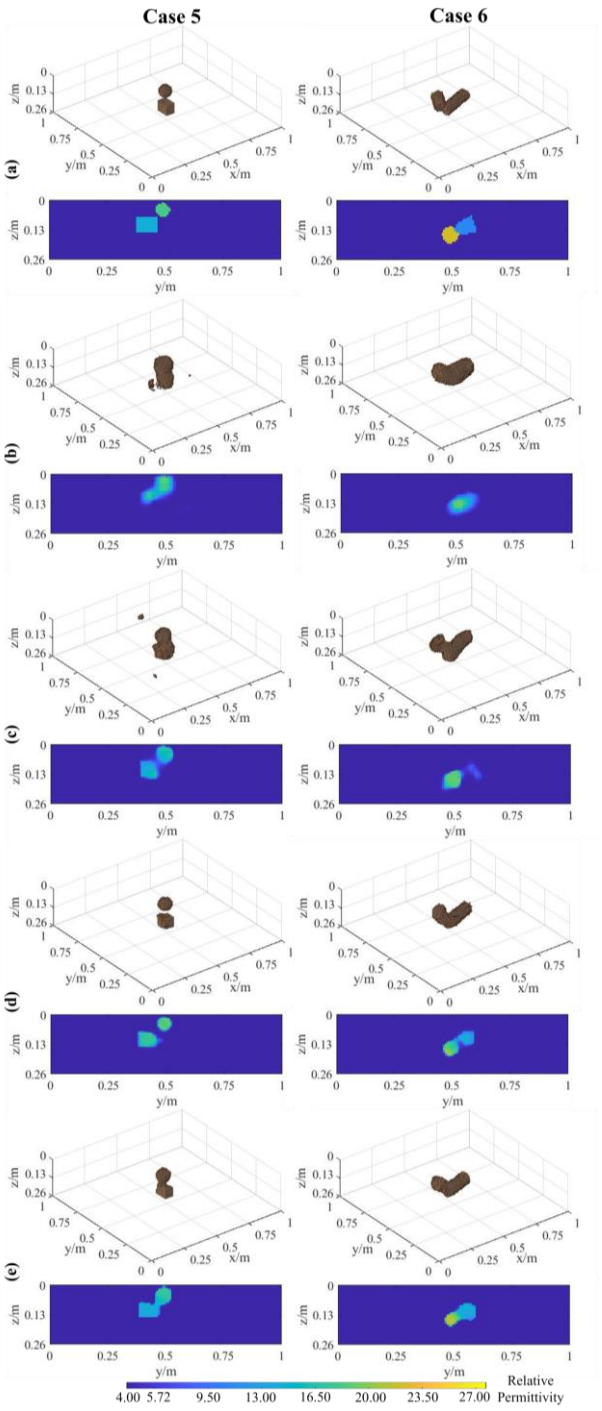


Figure 4.6: The imaging results of 3D permittivity maps for two-overlapping-object scenarios (Cases 5-6) in dataset I. While (a) shows the ground truths, (b)-(e) show the predictions using Models (1)-(4), respectively.

To further demonstrate the imaging performance for real simulated noisy data, Figure 4.7 presents the denoising and inversion results for dataset II. Figures 4.7(a)-(b) show the denoised C-scans and reconstructed 3D permittivity maps, respectively. Cases 7-9 provide three examples in groups i-iii, respectively. As shown, using the denoised C-scans with negligible clutter obtained from the Denoiser as inputs of the Inverter,

the predicted 3D permittivity maps match the ground truths. Although the two-object cases are not as precise as the results of dataset I using the networks trained with a large set of data, the subsurface objects can still be recognized well.

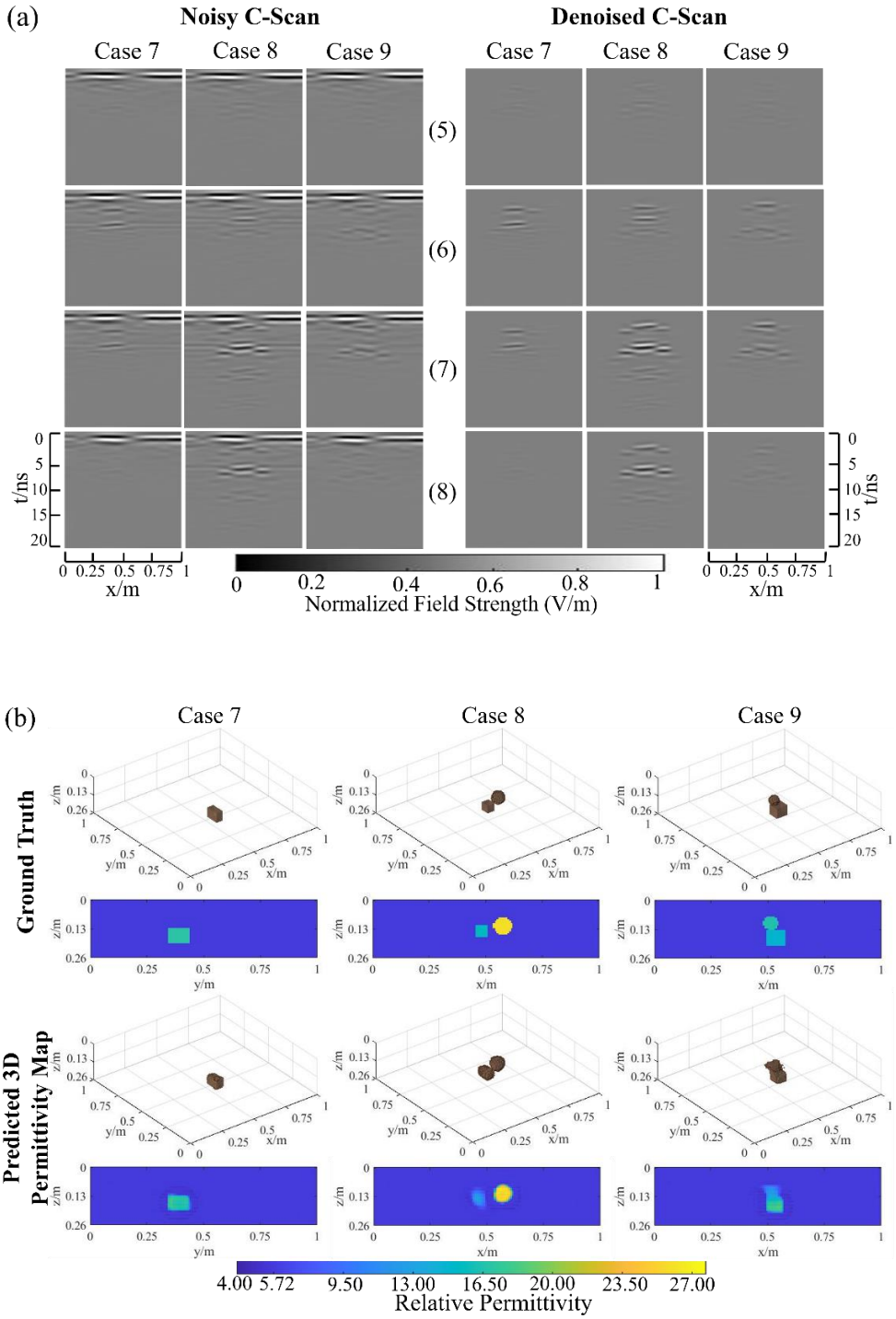


Figure 4.7: (a) Denoised C-scans obtained from Denoiser for dataset II. (b) Reconstructed 3D permittivity maps obtained from Invertor for dataset II. Cases 7-9 show the one-object scenario, two-separated-object scenario, and two-overlapping-object scenario, respectively.

4.3.4 Comparative Study

To prove the effectiveness of the proposed Denoiser and the MSFA modules in the Inverter, the reconstruction accuracy of four ablation models on dataset I is compared, as shown in Table 4.4. The four models include (1) the baseline 3D U-Net [121] with three successive convolutional layers in each encoding or decoding block, (2) only the Inverter, (3) the Denoiser and the baseline 3D U-Net [121], and (4) the proposed model.

Table 4.4: Inversion metrics comparison for the comparative study on dataset I.

Model	Denoiser	MSFA Module	SSIM ($\times 10^{-2}$) (↑)	MSE ($\times 10^{-2}$) (↓)	MAE ($\times 10^{-2}$) (↓)	MAPE (%) (↓)
(1)	×	×	99.06	14.12	2.72	0.46
(2)	×	✓	99.19	9.01	2.75	0.56
(3)	✓	×	99.55	9.32	1.92	0.36
(4)	✓	✓	**99.77**	**8.31**	**1.04**	**0.14**

As shown in Table 4.4, Model (1) performs the worst due to the interference of environmental noise in the permittivity reconstruction process and the limited perception view of the network. The performance of Model (2) is improved with the improved multi-scale perceptive capability introduced by the MSFA modules, but the performance is still limited by the interference of the environmental noise in C-scans. Compared with Model (1), Model (3) shows dramatically improved reconstruction accuracy with the noise suppression process in the Denoiser. Our proposed model (Model (4)) shows that first denoising the C-scan with Denoiser and then reconstructing permittivity maps with Inverter achieves the highest reconstruction accuracy.

The imaging results for one-object and two-separated-object scenarios in dataset I using the four models are compared in Figures 4.5(b)-(e), respectively, and those for two-overlapping-object scenarios are shown in Figures 4.6(b)-(e), respectively. It can

be observed that Model (1) performs the worst as shown in Figure 4.5(b) and Figure 4.6(b). The shapes and permittivity values of the reconstructed objects are far from their ground truths. As shown in Figure 4.5(c) and Figure 4.6(c), without the Denoiser, Model (2) incorrectly identifies the noise patterns as objects' reflections and reconstructs some clutters in the predicted 3D permittivity maps. As shown in Figure 4.5(d) and Figure 4.6(d), the clutters in the predicted maps using Model (3) are decreased due to the introduction of Denoiser, but the permittivity values and sizes of subsurface objects are not accurate especially for two-object scenarios as shown in Cases 3-6. This is due to the limited multi-scale feature extraction capability of the inversion network. As shown in Figure 4.5(e) and Figure 4.6(e), the reconstructed 3D permittivity maps using Model (4) that combines the Denoiser and MSFA modules in the Inverter are the closest to the ground truths regarding the sizes, locations, shapes, orientations, and permittivity values of the objects.

4.3.5 Hyperparameter Selection

Table 4.5: Evaluation metrics on dataset I using different hyperparameters.

Denoiser		SSIM ($\times10^{-2}$)	PSNR (dB)	MAE ($\times10^{-4}$)	RE (%)
m	C	(↑)	(↑)	(↓)	(↓)
0	8	99.63	53.04	9.43	0.45
1	8	99.88	57.86	6.02	0.26
2	**8**	**99.96**	**62.99**	**3.78**	**0.14**
2	4	99.81	55.67	7.88	0.34
2	2	98.75	46.57	13.86	0.95
Inverter		SSIM ($\times10^{-2}$)	MSE ($\times10^{-2}$)	MAE ($\times10^{-2}$)	MAPE (%)
n	$\tilde{C}$	(↑)	(↓)	(↓)	(↓)
2	8	98.23	10.67	3.67	0.75
3	8	98.93	7.77	2.96	0.61
4	**8**	**99.77**	**8.31**	**1.04**	**0.14**
4	4	99.63	8.50	1.54	0.24
4	2	99.65	9.77	1.65	0.26

To analyse the hyperparameter selection of the proposed scheme, the evaluation metrics of Denoiser and Inverter with different parameters are compared in Table 4.5.

As shown, for the Denoiser, a larger number of feature learning modules m and a higher channel number C lead to more accurate denoising results. For the Inverter, the reconstruction accuracy increases as the number of the encoding/decoding blocks n and the channel number $\tilde{C}$ increase. These performance improvements are due to the ability of the deepened and widened networks to describe more accurate mapping relationships between the input and output 3D images. However, the increase of these hyperparameters results in the increasing computational cost of the network training and testing, especially for the 3D case. According to the imaging results shown in Figures 4.3-4.6, when m and C of the Denoiser reach to 2 and 8, respectively, the denoised C-scans show clear reflection signatures of subsurface objects; when n and $\tilde{C}$ of the Inverter reach to 4 and 8, respectively, the reconstructed permittivity maps achieve high accuracy. Therefore, these hyperparameter values are selected to balance the imaging performance and the computational burden in our case.

4.4 Generalizability and Robustness Tests

4.4.1 Zero/Three-Object Scenarios

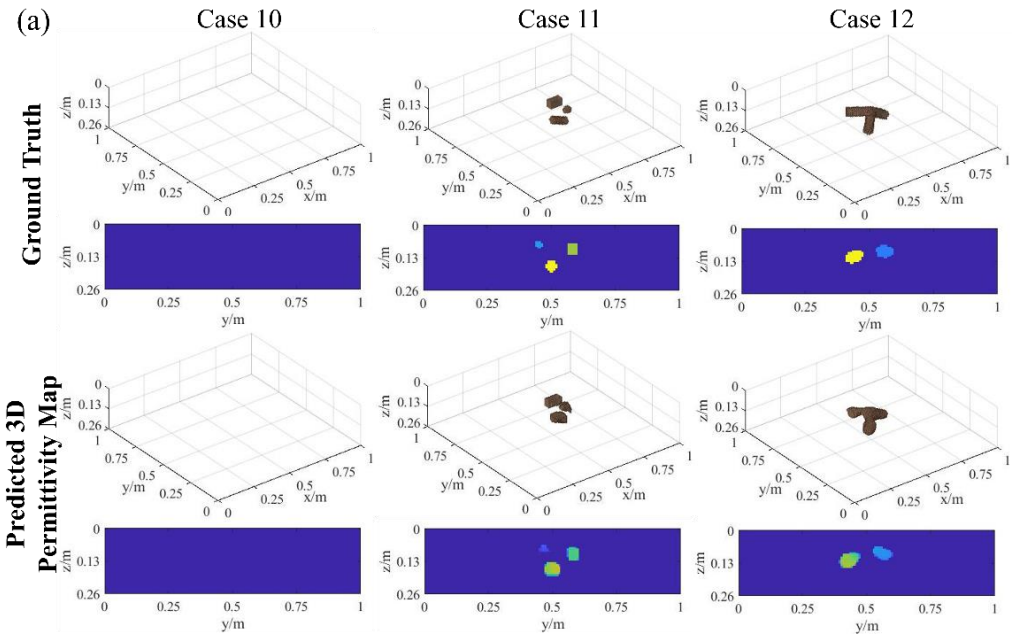
Dataset III. To evaluate the generalization capability of the proposed scheme, the pre-trained Denoiser and Inverter are employed to denoise C-scans and restore 15 scenarios with three objects or without any objects buried in the heterogeneous soil. It should be noted that only one-object and two-object scenarios are included in the pre-training dataset (dataset I), so the three-object cases and no-object cases are completely new scenarios to the denoising and inversion networks. Other properties of soil and objects and the dataset generation procedure are the same as those of dataset I.

Test Results. As shown in Table 4.6 (dataset III), without additional retraining or fine-tuning, the evaluation metrics of Denoiser and Inverter obtained for this testing dataset are still highly satisfactory. Using the denoised C-scans obtained from the Denoiser as inputs of the Inverter, the imaging results of reconstructed 3D permittivity maps are shown in Figure 4.8(a). Case 10 shows the imaging result when no object is buried in the soil. Both the reconstructed geometry and one-slice permittivity map contain only the soil environment. Cases 11 and 12 are the scenarios

when there are three buried objects with various random properties. It is clear from the 3D geometry and one-slice permittivity map that the predicted results match well with the ground truth. Although the training dataset does not cover the no-object and three-object scenarios, the proposed scheme can still reconstruct their 3D permittivity maps with good accuracy.

Table 4.6: Evaluation metrics of generalizability and robustness tests on datasets III-V.

Denoiser	SSIM (↑)	PSNR (dB) (↑)	MAE ($\times 10^{-4}$) (↓)	RE (%) (↓)
III	99.71	68.52	3.73	0.29
IV	99.99	69.97	1.41	0.07
V	99.97	60.26	5.15	0.20
Inverter	**SSIM ($\times 10^{-2}$) (↑)**	**MSE ($\times 10^{-2}$) (↓)**	**MAE ($\times 10^{-2}$) (↓)**	**MAPE (%) (↓)**
III	99.78	13.13	1.35	0.16
IV	98.97	13.10	2.82	0.35
V	98.68	9.33	3.31	0.66



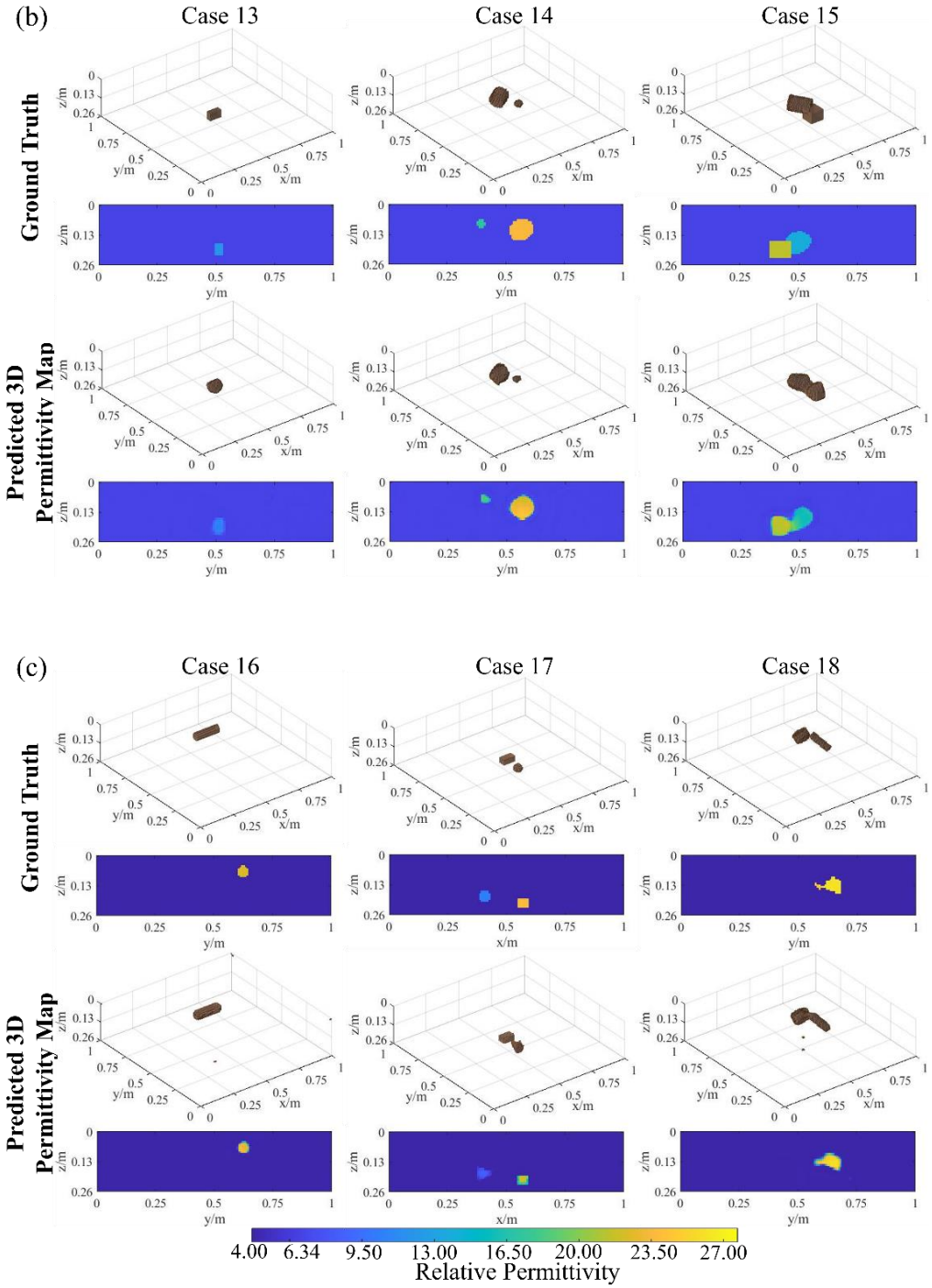


Figure 4.8: The imaging results of generalizability and robustness tests on (a) dataset III, (b) dataset IV, and (c) dataset V. Case 10 shows the scenario when no object is buried in the soil and Cases 11-12 are the scenarios when there are three buried objects with various random properties in dataset III. Cases 13-15 show the one-object, two-separated-object, and two-overlapping-object scenarios, respectively, in dataset IV. Cases 16-18 present the one-object, two-separated-object, and two-overlapping-object scenarios, respectively, in dataset V.

4.4.2 New Heterogeneous Soil Condition

Dataset IV. To evaluate the robustness of the proposed method in reconstructing subsurface scenarios from C-scans with new and different noise patterns, the 3D Peplinski mixing model with a new heterogeneous soil condition is constructed. Different from the soil properties in dataset II, the sand fraction, clay fraction, water content range, relative permittivity range, conductivity range are set as 0.5, 0.5, [0.1%, 15%], [3.62, 8.59], and [0.01, 0.06] S/m, respectively. The background permittivity in the ground-truth permittivity maps is set to 6.34, which is the average relative permittivity of the 50 materials in the new heterogeneous soil model. 285 additional training datasets are generated using the new 3D heterogeneous soil model to fine-tune the pre-trained network and 15 sets are produced for testing. The implementation details are the same as those for dataset II.

Test Results. As shown in Table 4.6 (dataset IV), the high SSIM and PSNR and low MAE and RE of the Denoiser indicate high denoising accuracy for the C-scans obtained under the new heterogeneous soil environment. Then the subsequent Inverter reconstructs 3D permittivity maps from denoised C-scans in high SSIM and low MSE, MAE, and MAPE. Cases 13-15 in Figure 4.8(b) show three imaging examples of the inversion results, including one-object, two-separated-object, and two-overlapping-object scenarios. It is clear from these visualized results that the predicted 3D permittivity maps are close to the ground truths, and the soil's permittivity and subsurface objects' properties are restored accurately.

4.4.3 Commercial GPR Antenna Model

Dataset V. To further test the generalization capability of the proposed scheme on a realistic commercial antenna, the GSSI 400-MHz GPR antenna model [125] is used in gprMax to transmit and receive the signals. Compared to the datasets I-IV obtained from the dipole/probe configuration at 1 GHz, both the operating frequency and the TX/RX configurations are the ones changed, and the height between the antenna model and ground is set to zero, resulting in highly different reflection patterns in C-scans. Other properties of subsurface scenarios are kept the same as those in dataset I. 285 datasets obtained with the GSSI 400-MHz GPR antenna model are produced to fine-tune the pre-trained networks. The fine-tuned networks are used to denoise 15 noisy C-scans and predict 3D permittivity maps.

Test Results. As shown in Table 4.6 (dataset V), the satisfactory denoising

performance with the commercial GSSI 400-MHz antenna model is demonstrated by the high SSIM and PSNR and low MAE and RE of the Denoiser. As for the inversion performance, the high SSIM and low MSE, MAE, and MAPE of the Inverter indicate good accuracy of subsurface 3D permittivity map reconstruction. To visually analyze the inversion results, the predicted 3D permittivity maps with the ground truths of the one-object, two-separated-object, and two-overlapping-object scenarios are compared, as shown in Cases 16-18 in Figure 4.8(c). It is clear from the imaging results that the predicted permittivity maps match well with their corresponding ground truths. The characteristics of the subsurface objects are restored with high accuracy. These results verify that using a small additional dataset to fine-tune the pre-trained networks yields an excellent generalization capability on a commercial antenna system with new operating frequencies and configurations.

4.5 Tests with Real Measurement Data

4.5.1 Dataset Collection and Experimental Setup

GPR experiments are carried out in a sandy test field to verify the applicability and effectiveness of the proposed scheme. As shown in Figure 4.9(a), the commercial GSSI's Utility Scan Pro GPR system with a 400-MHz antenna is used to obtain B-scans and then stack them into C-scans. The scanning area is 1×1 m^2. Every C-scan includes 21 B-scans and the distance between two adjacent traces is 5 cm. Every B-scan consists of 88 A-scans and the number of sampling points in every A-scan is 512. The time window is set as 20 ns. As shown in Figure 4.9(b), the scanning area is rotated to model various horizontal orientations of objects and shifted along the x and y axes to realize the scanning of the objects from various positions. Other properties, such as depth and vertical orientation, are changed by burying the objects. The buried objects are selected from five wooden objects with various shapes, sizes, and permittivity values. The ε_r values of the objects and the sand are measured by Keysight N1501A Dielectric Probe as shown in Figure 4.10(a). The average value of relative permittivities of five sand samples is 3.65, which is set as the background relative permittivity in the ground truths for training the proposed network. The properties of the buried objects in the experiments are shown in Figure 4.10(b). The ranges of the x-coordinate, y-coordinate, z-coordinate (depth), horizontal angle, and

vertical angle of the objects are [35, 65] cm, [35, 65] cm, [9, 25] cm, [0, 60] degree, and [0, 60] degree, respectively.

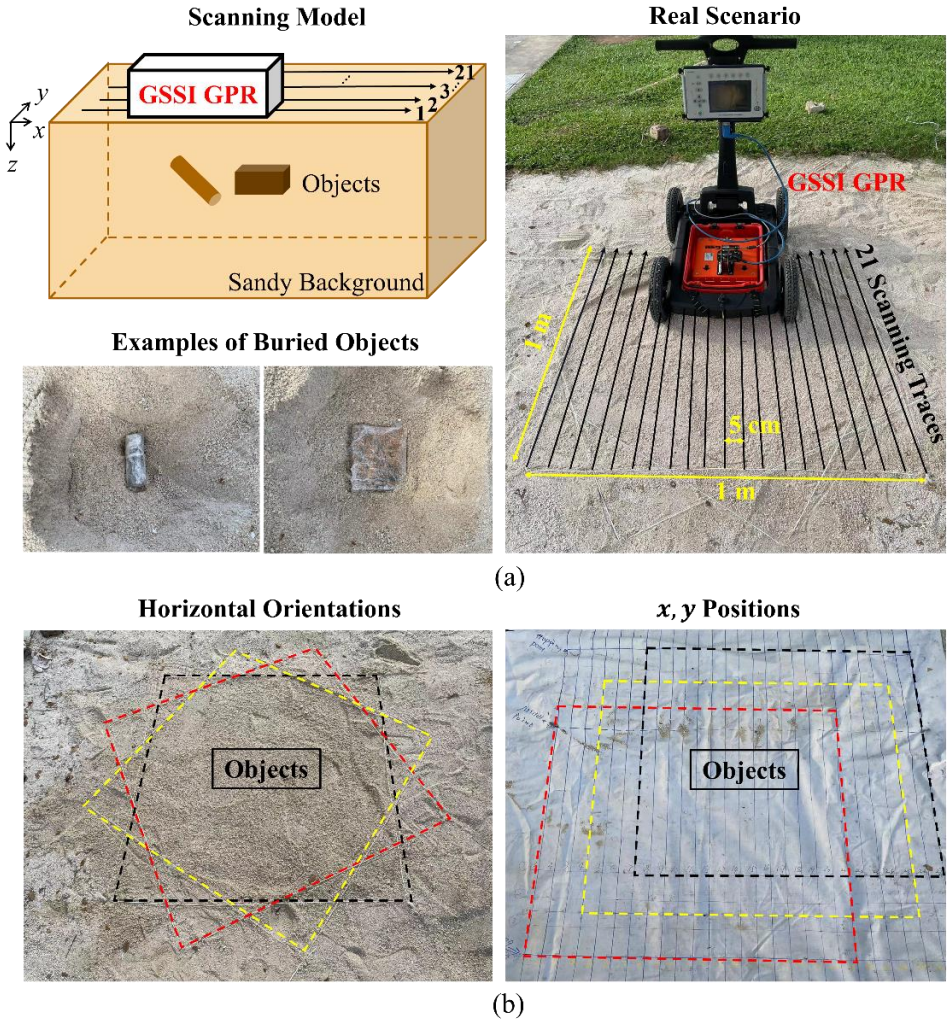


Figure 4.9: The real experimental scenario. (a) The scanning model with a commercial GPR system. (b) The scanning areas for modeling various horizontal orientations (left) and x, y positions (right).

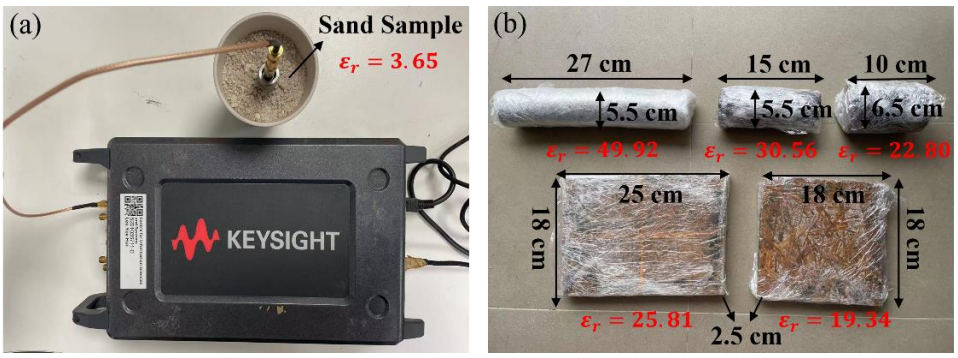


Figure 4.10: (a) The relative permittivity measurement setup. (b) The properties of buried objects utilized in real experiments. ε_r represents the relative permittivity.

Every measured C-scan is normalized to the range of [0, 1], resized to 128×128×128, and pre-processed by time-zero correction and mean subtraction to form the input noisy C-scan. To obtain the ground-truth denoised C-scan, a C-scan is measured in an empty area far away from the object region and subtracted from the noisy C-scan. A total of 220 sets of real measured noisy C-scans, denoised C-scans, and the subsurface 3D permittivity maps are obtained to fine-tune the pre-trained Denoiser and Inverter for 100 epochs.

4.5.2 Inversion Result Analysis

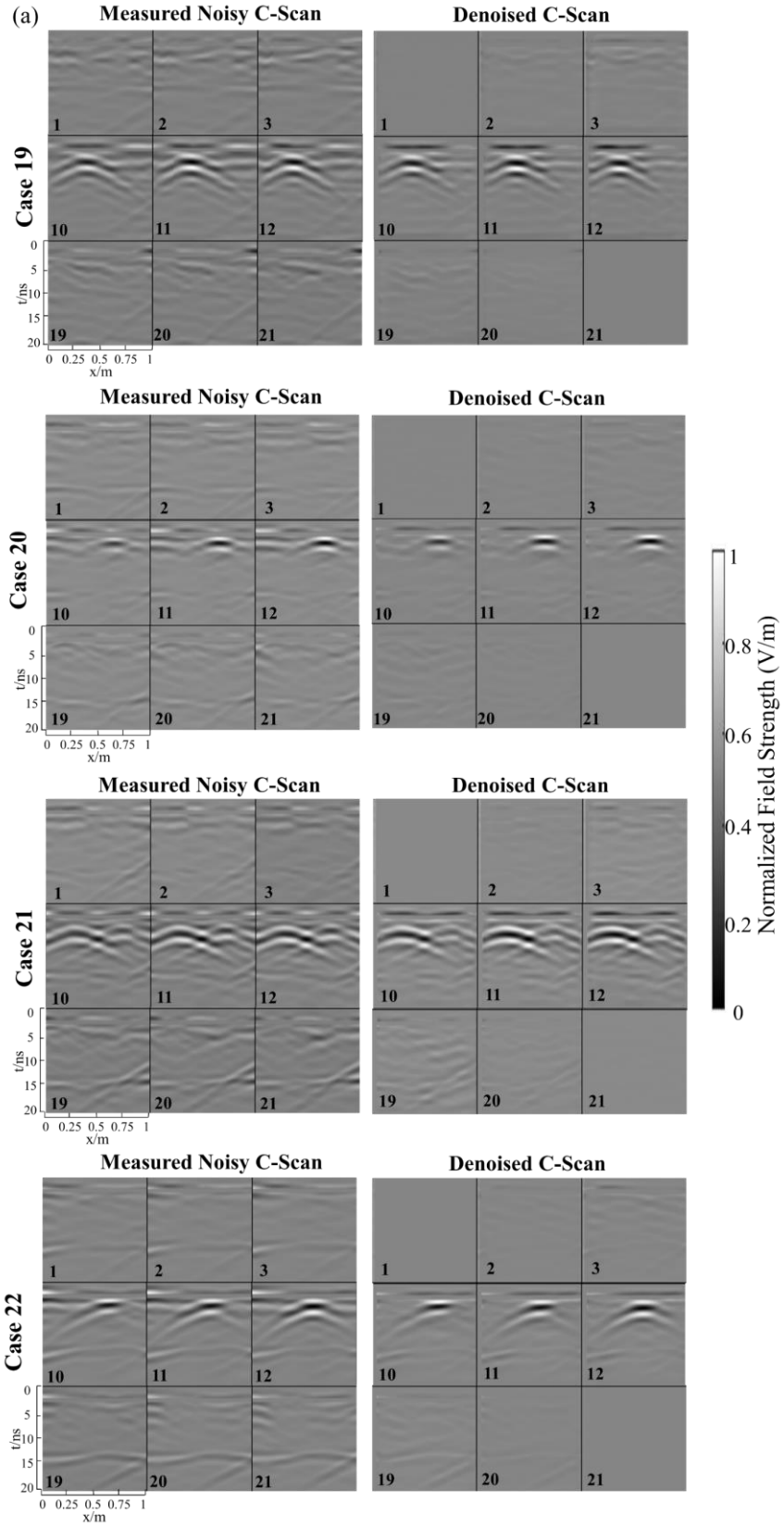
Table 4.7: Evaluation metrics of real measured testing data.

Groups	Testing Data No.	Denoiser				Inverter			
		SSIM ($\times10^{-2}$) (↑)	PSNR (dB) (↑)	MAE ($\times10^{-4}$) (↓)	RE (%) (↓)	SSIM ($\times10^{-2}$) (↑)	MSE ($\times10^{-1}$) (↓)	MAE ($\times10^{-2}$) (↓)	MAPE (%) (↓)
i	(1)	99.95	58.72	7.78	0.23	99.64	4.89	3.38	0.61
	(2)	99.96	59.96	6.94	0.20	99.60	8.52	4.31	0.58
	(3)	99.96	59.31	7.05	0.22	99.76	1.73	2.26	0.43
	(4)	99.96	59.27	7.51	0.22	99.70	2.50	2.73	0.37
	(5)	99.95	59.12	6.87	0.22	99.61	1.25	2.69	0.49
	Ave.	**99.96**	**59.28**	**7.23**	**0.22**	**99.66**	**3.78**	**3.08**	**0.49**
ii	(6)	99.94	57.15	9.66	0.28	99.57	6.58	4.15	0.48
	(7)	99.94	57.15	9.17	0.27	99.57	3.54	3.85	0.66
	(8)	99.94	57.71	8.58	0.27	99.48	8.57	5.14	0.82
	Ave.	**99.94**	**57.34**	**9.14**	**0.27**	**99.54**	**6.23**	**4.38**	**0.65**
iii	(9)	99.96	59.43	7.16	0.21	99.44	8.81	5.59	0.80
	(10)	99.95	59.29	7.27	0.22	99.37	11.37	5.79	0.89
	Ave.	**99.96**	**59.36**	**7.22**	**0.22**	**99.41**	**10.09**	**5.69**	**0.85**
i, ii, and iii	Ave.	**99.95**	**58.75**	**7.80**	**0.23**	**99.57**	**5.78**	**3.99**	**0.61**

Quantitative results. Using the fine-tuned networks, 10 measured testing datasets are used to evaluate the performance of the proposed scheme. The quantitative results of the Denoiser and Inverter are shown in Table 4.7. Groups i, ii, and iii include 5 one-object cases, 3 two-separated-object cases, and 2 two-touching-object cases, respectively. For the Denoiser, the average SSIM, PSNR, MAE, and RE have an acceptable degradation compared to the previous numerical datasets. The accuracy degradation is reasonable as the real measured C-scans, as shown in Figure 4.10(a), have randomly distributed noisy patterns interfering with the objects' reflection signatures. With the denoised C-scans, the reconstruction accuracy of the Inverter for various subsurface scenarios is evaluated in Table 4.7. It can be observed that the MSE, MAE, and MAPE of group i are the lowest and the SSIM of group i is the highest, while the four metrics of groups ii and iii are slightly worse than those of group i as the signatures of two objects in the obtained C-scan are more complicated. When two objects are in touch in group iii, their hyperbolic patterns are highly interleaved, resulting in a further decrease in reconstruction accuracy.

Imaging results. The imaging results of denoised C-scans and reconstructed 3D permittivity maps are shown in Figure 4.11. Cases 19-20 are one-object scenarios, Case 21 is a two-separated-object scenario, and Case 22 is a two-touching-object scenario. Figure 4.11(a) presents the input noisy C-scans and the denoised C-scans obtained from the Denoiser. Selected B-scans that include the main reflection signatures or obvious noise patterns in each C-scan are visually presented. Compared to the noisy C-scans, the denoised ones contain less noise and clutter due to reflections from the subsurface environment, and the objects' reflection patterns are clearer. Figure 4.11(b) shows the 3D reconstruction results predicted from the denoised C-scans using the Inverter. The predicted 3D permittivity maps and their ground truths, including the 3D geometries and one-slice of the permittivity maps, are compared. Cases 19 and 20 show the reconstruction results for a cylinder and a box, respectively. The results predicted by the Inverter match well with the ground truth ones. As for the two-separated-object scenario and the two-touching-object scenario shown in Cases 21 and 22, the reconstructed 3D permittivity maps are close to the ground truth ones in terms of the shapes, sizes, orientations, positions, and permittivity values of the subsurface objects. Although some clutter appears around the objects and the edge between the two touching objects is slightly blurry, the properties of the subsurface objects can be clearly observed. These quantitative and qualitative results

demonstrate the applicability of the proposed 3DInvNet in effectively reconstructing the subsurface 3D permittivity maps from real measurement data.



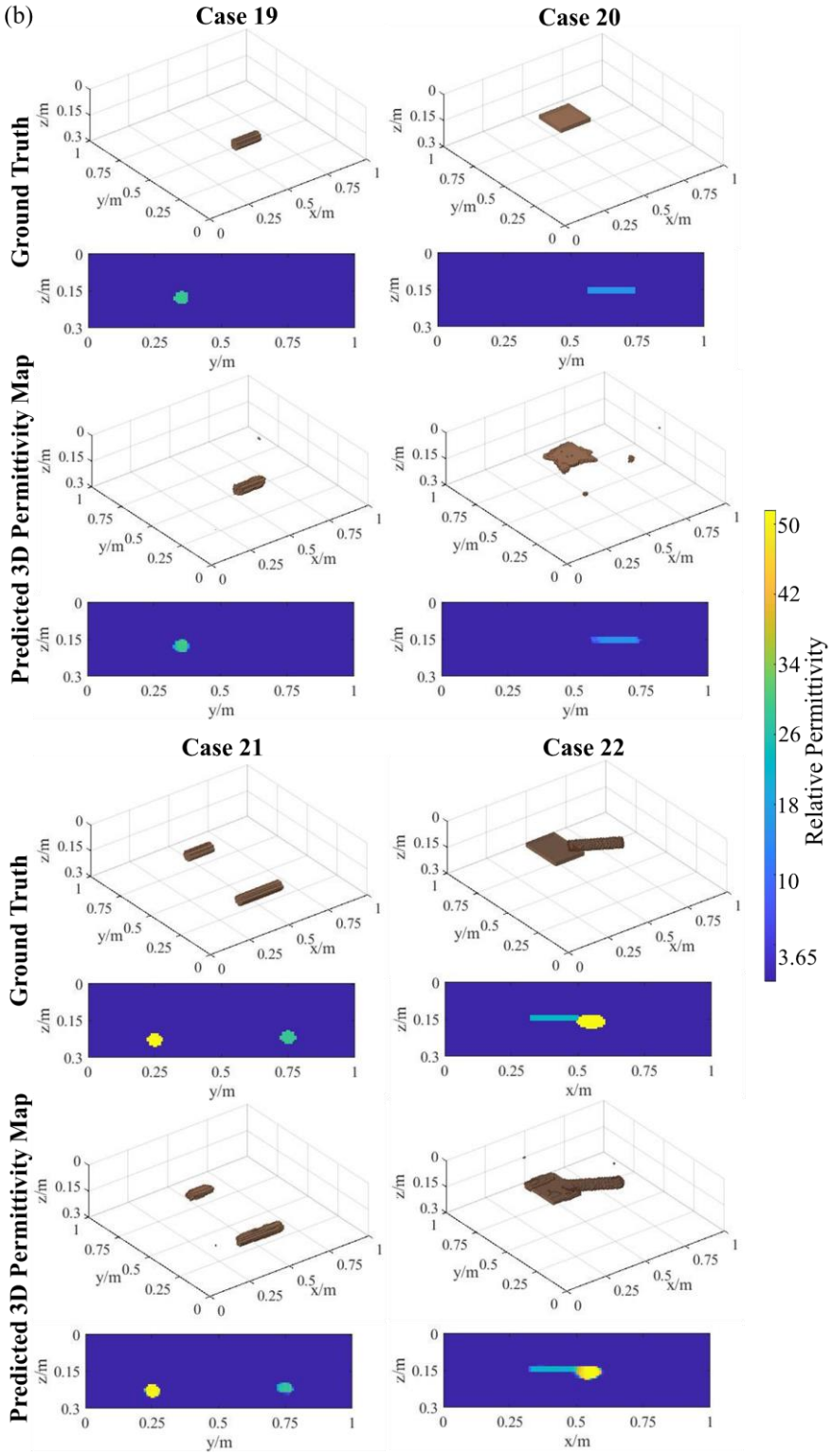


Figure 4.11: (a) The 3D denoising results for real measured C-scans. (b) The 3D permittivity map reconstruction results. Cases 19-20 are one-object scenarios, Case 21 is a two-separated-object scenario, and Case 22 is a two-touching-object scenario.

4.6 Conclusion

In this work, a two-stage scheme called 3DInvNet was proposed to rapidly denoise GPR C-scans and reconstruct 3D subsurface permittivity maps. The first stage, called Denoiser, effectively extracts the informative features of subsurface objects and suppresses the environmental noise of the GPR C-scans. The second stage, called Inverter, builds the relationship between the discriminative features of the denoised C-scans with the corresponding subsurface permittivity maps. The test results on both the numerical and real measurement data demonstrate that the proposed method can accurately and efficiently reconstruct 3D permittivity maps, including the objects' shapes, sizes, positions, and permittivity values. Moreover, the generalization capability and robustness of the proposed network were demonstrated by its application to various subsurface scenarios. This work was the first that implemented a 3D deep learning technique for 3D subsurface structure restoration. It provides a novel and efficient method to solve the GPR 3D inverse problem and alleviate the computational burden of 3D reconstruction.

Chapter 5

Fast GPR Forward Solver

To alleviate the computational burden of large dataset generation for training the learning-based inversion algorithms, this chapter introduces a deep learning-based GPR forward solver to rapidly predict the GPR B-scan for given permittivity and conductivity maps of the subsurface scenario. The introduction, methodology explaining the designed network and learning process, numerical experiments, and comparative results are provided as follows.

5.1 Introduction

The forward full-wave modeling of the GPR system is of great significance for understanding the subsurface scattering mechanisms and interpretation and inversion of the GPR data. In our proposed deep learning-based inversion schemes, large sets of numerical simulation data are required to pre-train the deep learning models. However, as reviewed in Section 2.3, traditional physics-based forward solvers [74-78] require excessive computational time, especially when their repetitive executions are required by signal processing or deep learning-based inversion algorithms. Using a traditional physics-based forward solver, the dataset generation becomes highly time-consuming. To alleviate the computational burden of traditional GPR forward solvers, machine learning-based fast forward solvers have been proposed in [79-82]. Nevertheless, all these learning-based studies are limited in two aspects. First, they explore the parameter-to-parameter relationship between the input parameters related to the GPR scenario and the output parameters, which only yield partial information about the GPR scenario. For example, the output A-scan in [81-82] only depicts the reflected signal at one position, which only contains information in the narrow spatial domain that affects the signal. It cannot capture the reflected signals at different positions that contain more object information. Furthermore, the applied subsurface scenarios are simple and limited in the training domain. So far, there exists no study exploiting deep learning algorithms for GPR forward modelling and forming a

mapping relationship between the input image of the GPR scenario and the output image (e.g., B-scan) that can provide the complete 2D information of the subsurface scenario.

The main challenge of developing a deep learning-based GPR forward solver is its high dependency on a large set of training data due to its data-driven characteristics. When faced with new scenarios that are out of the training data domain, the solver will become ineffective as it has not learnt the knowledge of the new scenarios in the training process. One possible solution to this issue is the transfer learning technique [98], in which the knowledge learned from a task with a lot of available labeled training data in the source domain is re-used in a new related task that doesn't have much data in the target domain in order to boost performance on the new task. Fine-tuning is a common approach of transfer learning, including fine-tuning the network structure, model parameters, and/or learning strategy. There were many studies that involve various fine-tuning methods for transfer learning in the deep learning field [126-129]. In [126], using the pre-trained model as a constraint to guide the fine-tuning was proposed to improve the robustness of the fine-tuned model via adding a regularization loss between the pre-trained model parameters and fine-tuned model parameters. In [127], a method of dynamic pruning was proposed to realize that the fine-tuning stage adjusts not only the model parameters but also the model network structure. In [128], an input-dependent fine-tuning approach was proposed to automatically determine which layers to fine-tune per instance in the target domain. In [129], filter selection, layer-wise recurrent gated network, and gated batch normalization techniques were proposed to allow different images in the target dataset to fine-tune different convolutional filters in the pre-trained model. These studies have verified the effectiveness of fine-tuning for boosting the performance of the new target task.

In this work, we propose a deep learning-based GPR forward solver for characterizing 2D radar signatures of subsurface convex objects. A deep bimodal encoder-decoder neural network is designed to predict the GPR B-scan for given permittivity and conductivity maps of the subsurface scenario under heterogenous soil conditions. Two independent bimodal encoders extract informative features from input permittivity and conductivity maps. A feature fusion module with cross-attention is employed to adaptively fuse the extracted bimodal feature representations. The

decoder subsequently constructs the B-scans from the fused feature representations. Transfer learning is introduced to boost the network's generalization capability for new scenarios vastly different from those in the training set. The test results show that the average RE achieved by the proposed solver reaches 1.28% and the computation time for predicting one B-scan is only 12 milliseconds.

5.2 Methodology

Let x and y denote a subsurface scenario and its corresponding B-scan obtained via GPR scanning. A deep encoder-decoder neural network is proposed to form the mapping $h(\cdot)$ from x to y. The B-scans obtained from an FDTD solver are used to train the network. Using the well-trained network, the B-scans can be automatically and rapidly predicted from given subsurface scenarios via $y = h(x)$. To improve the applicability to complex scenarios, we take into account the subsurface permittivity and conductivity distributions under heterogeneous soil conditions as shown in Figure 5.1. The relative permittivity (ε_r) map x_{ε_r} and conductivity (σ) map x_σ of the subsurface scenario x are set as the inputs of the network. The output of the network is the corresponding B-scan via $y = h(x_{\varepsilon_r}, x_\sigma)$. The detailed network structure and learning method are described as follows.

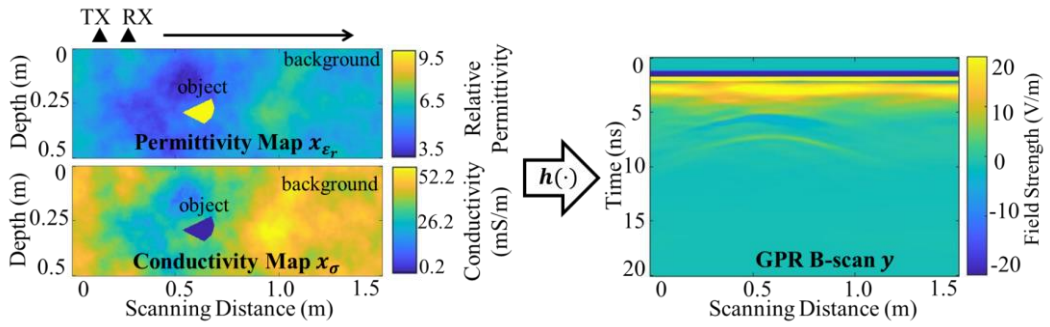


Figure 5.1: An example of the GPR forward problem under heterogeneous background conditions. TX and RX denote the transmitter and the receiver in the GPR survey. $h(\cdot)$ represents the mapping from the relative permittivity map x_{ε_r} and the conductivity map x_σ of the subsurface scenario to the corresponding B-scan y obtained via GPR scanning.

5.2.1 Bimodal Encoder

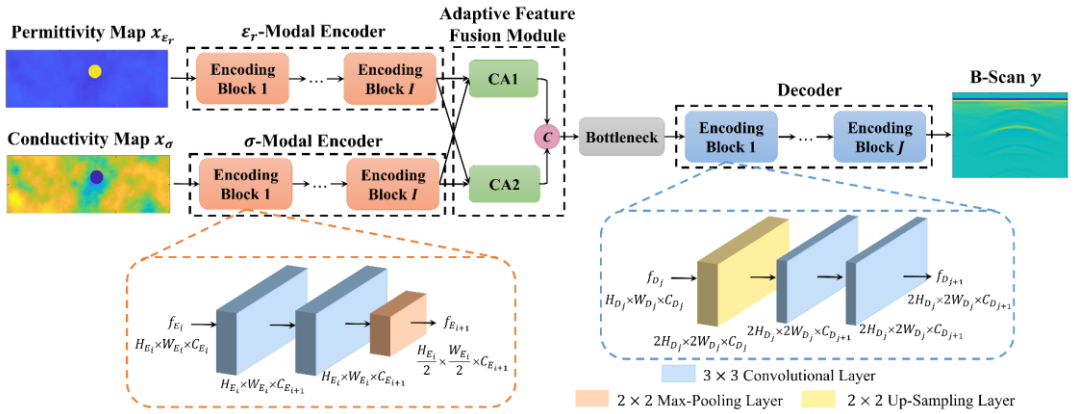


Figure 5.2: The structure of the proposed network. i, j and I, J are the indices and total numbers of encoding and decoding blocks, respectively. "c" in the pink circle represents the operation of concatenation. f_{E_i} and f_{D_j} represent the feature maps of the i^{th} encoding block and the j^{th} decoding block in the encoder and decoder, respectively. $H_{E_i}, W_{E_i}, C_{E_i}$ and $H_{D_j}, W_{D_j}, C_{D_j}$ are the heights, widths, and channel numbers in the encoder and decoder, respectively. "CA1" and "CA2" represent the permittivity-modal-guided cross-attention block and the conductivity-modal-guided cross-attention block, respectively.

The whole structure of the proposed network is shown in Figure 5.2. The left path of the network includes a ε_r-modal encoder and a σ-modal encoder, which independently extract informative features from the input permittivity map x_{ε_r} and conductivity map x_σ. Each encoder consists of I consecutive encoding blocks. Every encoding block is made up of two consecutive 3×3 convolutional layers with the strides of 1×1, followed by a 2×2 max-pooling layer with the stride of 2×2 for down-sampling. The ReLU activation follows every convolutional layer to provide high non-linearity of the network. Assume the dimension of the input feature map f_{E_i} in the i^{th} encoding block is $H_{E_i} \times W_{E_i} \times C_{E_i}$, the output feature map $f_{E_{i+1}}$ becomes $\frac{H_{E_i}}{2} \times \frac{W_{E_i}}{2} \times C_{E_{i+1}}$. Finally, a ε_r-modal feature representation F_{ε_r} and a σ-modal feature representation F_σ are obtained from the two encoders, respectively.

5.2.2 Adaptive Feature Fusion Module

To enhance the connectivity between F_{ε_r} and F_σ and capture cross-modal information, an adaptive feature fusion module is designed as shown in Figure 5.3. In this module, the inputs of a self-attention block [130] are the vectors of query Q, key K, and value V, projected by a linear feed-forward layer (the 1×1 convolution).

The output attention feature A is produced by weighted summation over V as formulated in Equation (5.1). The dot products of Q and K are computed, scaled by $\sqrt{d}$ (d is the dimension of K), and followed by a Softmax function to obtain the weights on V.

$$A(Q,K,V) = Softmax\left(\frac{Q \cdot K^T}{\sqrt{d}}\right) \cdot V \tag{5.1}$$

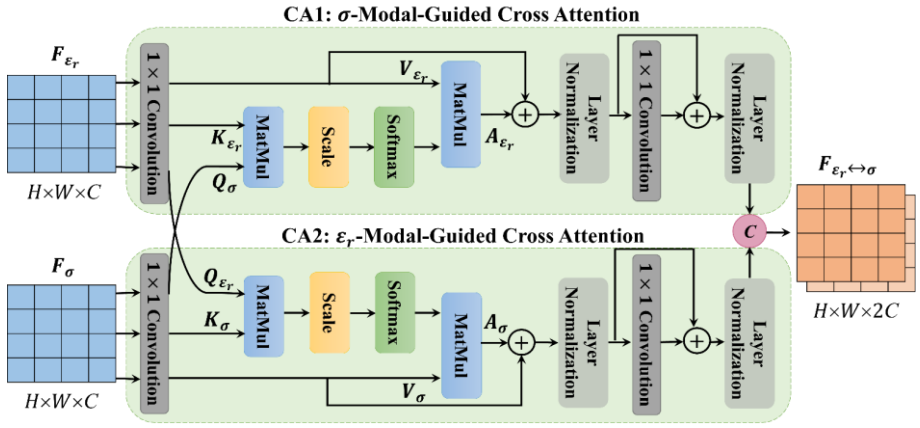


Figure 5.3: The structure of the adaptive feature fusion module. ε_r and σ represent the relative permittivity and conductivity, respectively. $F_{\varepsilon_r}$and F_σ represent the ε_r-modal feature representation and the σ-modal feature representation, respectively. K_{ε_r}, Q_{ε_r}, and V_{ε_r} are the key, query, and value of F_{ε_r}, respectively. K_σ, Q_σ, and V_σ are the key, query, and value of F_σ, respectively. A_{ε_r} and A_σ represent the ε_r-modal and σ-modal attention features, respectively. "c" in the pink circle represents the operation of concatenation. $F_{\varepsilon_r \leftrightarrow \sigma}$ is the fused feature representation with cross-modal information.

To adaptively fuse the bimodal features in F_{ε_r} and F_σ, two cross-attention (CA) blocks [131-132], CA1 and CA2, are employed. The query of F_{ε_r} (Q_{ε_r}) and that of F_σ (Q_σ) are exchanged with each other, which introduces cross-modal interactions between the ε_r-modal and σ-modal features. The A_{ε_r} in CA1, which represents the ε_r-modal attention features obtained with the guidance of σ-modal information, and the A_σ in CA2, which represents the σ-modal attention features obtained with the guidance of ε_r-modal information, are computed by

$$A_{\varepsilon_r}(Q_\sigma, K_{\varepsilon_r}, V_{\varepsilon_r}) = Softmax\left(\frac{Q_\sigma \cdot {K_{\varepsilon_r}}^T}{\sqrt{d}}\right) \cdot V_{\varepsilon_r}, \tag{5.2}$$

$$A_\sigma\left(Q_{\varepsilon_r}, K_\sigma, V_\sigma\right) = Softmax\left(\frac{Q_{\varepsilon_r} \cdot {K_\sigma}^T}{\sqrt{d}}\right) \cdot V_\sigma. \tag{5.3}$$

To jointly learn the information from different representation subspaces, multi-head attention [130-132] structure is adopted in both CA1 and CA2. The functions (5.2) and (5.3) are performed on m paralleled heads that are linearly projected from Q, K, and V. The output features of each head are concatenated and linearly projected to A, which is once again followed by 1×1 convolution. Residual connection and layer normalization are applied in each sub-layer. Then two extracted feature representations with cross-modal information are concatenated as the fused feature representation $F_{\varepsilon_r \leftrightarrow \sigma}$.

5.2.3 Decoder and Loss Function

A bottleneck block consisting of a 3×3 convolutional layer and a 3×3 transposed convolutional layer follows to bridge the obtained $F_{\varepsilon_r \leftrightarrow \sigma}$ and the decoder. The decoder includes J repeated decoding blocks to predict the GPR B-scan. As shown in Figure 5.2, every decoding block has one 2×2 up-sampling layer with the stride of 2×2 and two 3×3 convolutional layers with the strides of 1×1. In the final stage, a 1×1 convolutional layer with the stride of 1×1 followed by a linear activation outputs the B-scan y. To train the proposed network with a fast convergence rate and effectively suppress large errors, the MSE between the B-scan predicted by the proposed network $h_\theta(\cdot)$ and the ground truth B-scan $\hat{y}$ generated by the physics-based solver is adopted as the loss function; here θ represents the trainable parameters of the network. The objective function is expressed as

$$\min_\theta \left(h_\theta\left(x_{\varepsilon_r}, x_\sigma\right) - \hat{y}\right)^2. \tag{5.4}$$

5.2.4 Fine-Tuning for New Scenarios

One limitation of applying deep learning-based data-driven techniques to EM forward modeling problems is their high dependency on the training data. When the testing data is vastly different from the training data, the network becomes ineffective. To tackle this issue, fine-tuning developed for transfer learning is introduced to predict

the B-scans of subsurface objects in new scenarios. In the fine-tuning process, we retain the network structure but set the whole pre-trained model for the source domain as the starting point of the model training for the new scenarios in the target domain. In particular, first, the proposed network is pre-trained with a large and diverse dataset. It is noted that the pre-training only needs to be done once. Second, using the physics-based solver, a small additional training data is generated for new scenarios out of the pre-training set. Third, the pre-trained network is set as the initial state and is further optimized based on Equation (5.4) using new training data until convergence. The learning rate should be lower than the one in the pre-training process to avoid exploding gradient issues. Finally, the fine-tuned network with enhanced generalization capability can accurately predict the B-scans for new scenarios.

5.3 Numerical Experiments

5.3.1 Dataset Generation and Network Training

To train and test the proposed network, a diverse set of data is generated using an open-source FDTD-based software gprMax [65-66] executed on a NVIDIA Quadro RTX8000 GPU. In the simulation scenario, as shown in Figure 5.1, a 2D domain with dimensions of 1.5×0.5 m^2 is discretized by pixels with dimensions of 0.0025×0.0025 m^2. The time window is set as 20 ns. A Hertzian dipole (TX) transmitting a Gaussian waveform with a center frequency of 1 GHz is used. A probe (RX) positioned 20 cm away from TX receives the fields. The TX and RX in a common-offset mode move along one straight scanning trace to obtain the B-scan. The scanning step is 2.5 cm. Convex objects with six shapes, including the circle, ellipse, hexagon, pentagon, quadrilateral, and triangle, are considered as subsurface objects. Each object has a random size, position, relative permittivity, and conductivity. The Peplinski mixing model [70-72] is used to form a heterogeneous soil environment with realistic dielectric and geometric properties. The soil properties are set as follows: sand fraction 0.5, clay fraction 0.5, bulk density 2 g/cm^3, and sand particle density 2.66 g/cm^3. 50 different materials over a range of water content from 0.1% to 10%, described by 50 Debye functions, are randomly distributed. The ε_r and σ of the background soil medium vary within [3.59, 7.17] and [8.37, 52.13] mS/m,

respectively, while ε_r and σ of the object are randomly selected from [1, 32] and [0, 0.8] mS/m, respectively. Ten random distributions are generated to model 10 diverse heterogeneous soil environments. In total, 15,000 sets of $[x_{\varepsilon_r}, x_\sigma, \hat{y}]$ are generated to train the proposed network, and 1,500 sets are generated to test the performance.

The proposed network implemented on TensorFlow is trained using the obtained dataset. For this purpose, Adam optimizer is used to minimize the loss function (Equation (5.4)). Five encoding blocks and five symmetric decoding blocks are employed. The $[C_{E_1}, \dots, C_{E_5}]$ and $[C_{D_1}, \dots, C_{D_5}]$ are set as [16, 32, 64, 128, 256] and [256, 128, 64, 32, 16], respectively. m and d are set as 8 and 32, respectively, in the multi-head attention structure. After 100-epoch training, the model with the lowest testing loss predicts y for the given x_{ε_r} and x_σ.

5.3.2 The Analysis of Regular Testing Results

To quantitatively evaluate the performance, the RE between the predicted result y and ground truth B-scan $\hat{y}$ is defined as [133]

$$RE(y, \hat{y}) = \frac{\sqrt{\sum_{i=1}^{H}\sum_{j=1}^{W}\left(y_{i,j} - \hat{y}_{i,j}\right)^2}}{\sqrt{\sum_{i=1}^{H}\sum_{j=1}^{W}\hat{y}_{i,j}^2}} \cdot 100\%, \tag{5.5}$$

Table 5.1 compares the average RE of the 1,500 regular testing data using three networks when (i) the network adopts a baseline encoder-decoder structure, in which a single encoder takes the concatenated x_{ε_r} and x_σ as a two-channel input, (ii) the bimodal encoder replaces the single encoder, while the output F_{ε_r} and F_σ are concatenated directly, and (iii) the final network includes the adaptive feature fusion module and the bimodal encoder. As shown, the RE of the regular data achieved by the baseline network (i) is the largest, while that achieved by the network (ii) is relatively smaller compared to that of network (i) due to the introduction of the bimodal encoder. The addition of the adaptive feature fusion module in network (iii) yields a further decrease in RE, as it enhances the cross-modal connectivity and captures more informative features from x_{ε_r} and x_σ. The comparative results demonstrate the effectiveness of the proposed bimodal encoder and adaptive feature fusion module. The low RE (1.28%) of the final network (iii) shows that the predicted

B-scans using the proposed method are highly close to the ones obtained by the FDTD solver. By comparing the parameters, training time, and testing time of three networks, the computational cost of network (iii) is higher than those of networks (i) and (ii). However, the computation time for obtaining one B-scan by network (iii) is only 12 milliseconds, whereas the FDTD solver requires 4.5 minutes to obtain one B-scan on the same GPU. Thereby, the speed-up achieved by network (iii) is as large as 22,500x.

To qualitatively analyse the performance, the predicted B-scans using the proposed network are compared with the ones obtained by the FDTD solver. Cases 1-6 in Figure 5.4 present the comparative results for the subsurface object with various shapes, sizes, locations, permittivity values, and conductivities. For the given permittivity and conductivity maps under various heterogeneous background soil distributions, the B-scans predicted by the proposed network match well with the B-scans obtained by the FDTD solver. Furthermore, the absolute errors between two sets of B-scans are too small and cannot be observed by human eyes.

Table 5.1: Qualitative accuracy and efficiency comparison.

Networks	Parameters	Training Time (h)	Testing Time (ms)	Speed-Up	RE	
					Regular Data	Generalized Data
(i)	9,436,401	2.94	8	33,750	1.55%	4.44%
(ii)	11,794,385	3.05	9	30,000	1.41%	4.37%
(iii)	12,717,521	3.14	12	22,500	**1.28%**	**3.57%**

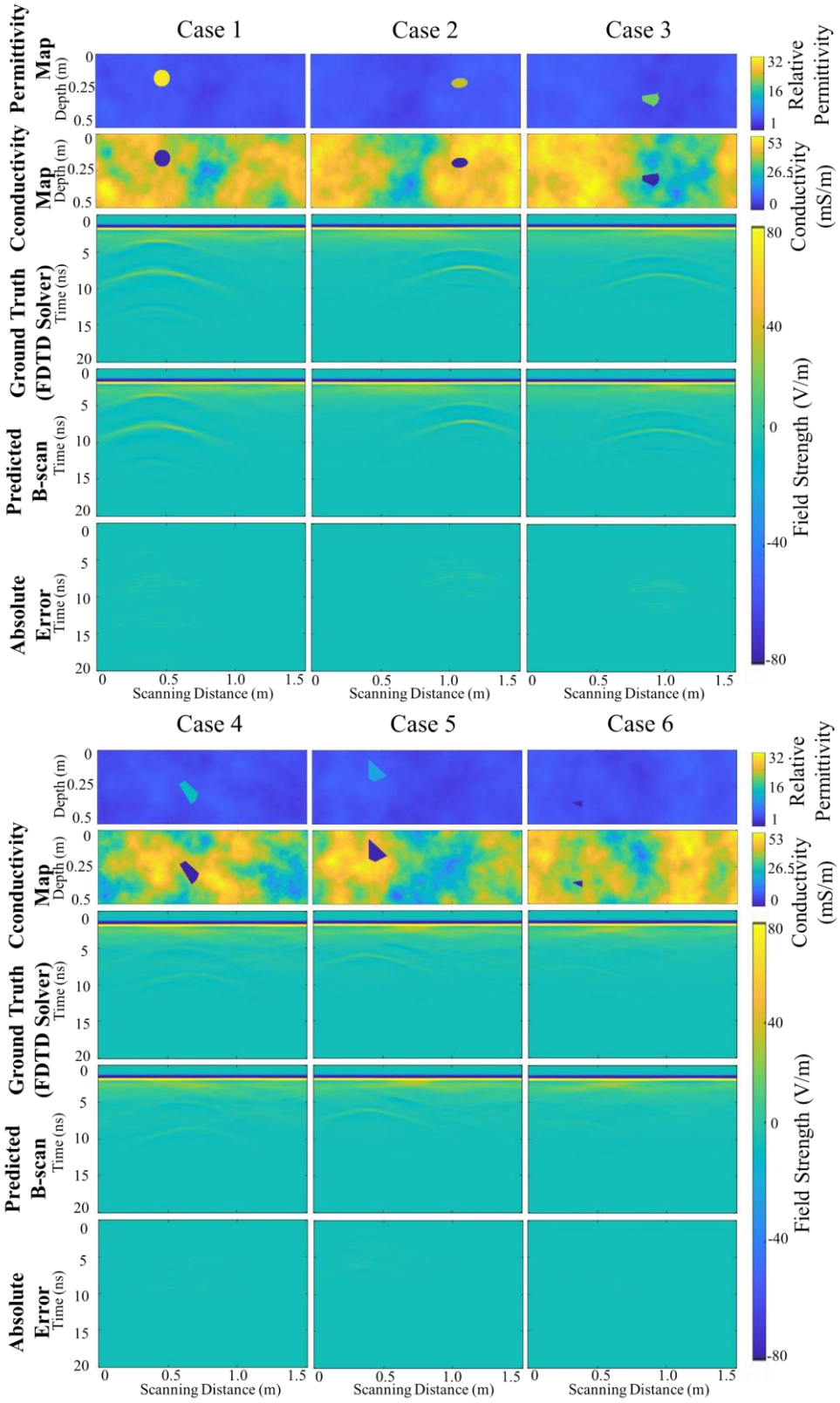


Figure 5.4: The comparison of the test results. Ground truth B-scans are obtained by the FDTD solver while the predicted B-scans are the ones obtained using the proposed network. The absolute error figures show the pixel-wise absolute difference between the ground truth and the predicted B-scans.

5.3.3 Generalizability Tests

1) Random Convex Objects

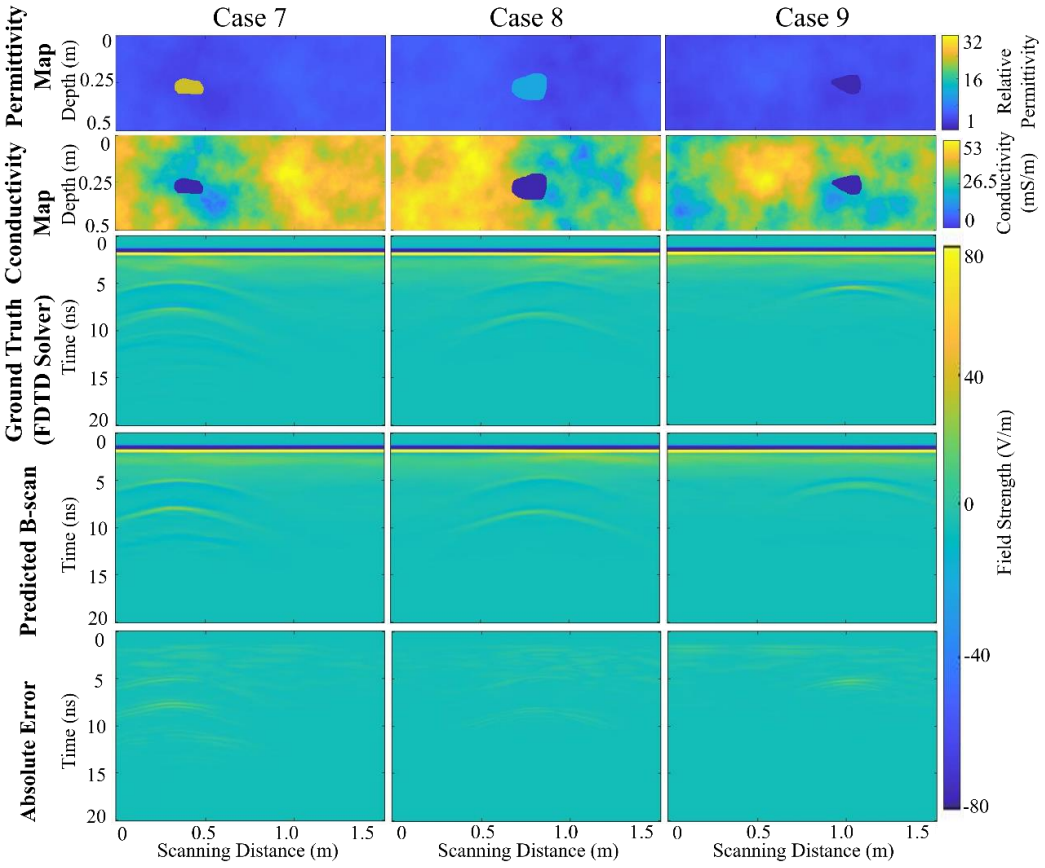


Figure 5.5: The tests on the generalized data realized by hand-drawn convex objects.

To test the generalization capability of the proposed network for convex objects with random shapes, three sets of permittivity and conductivity maps with hand-drawn convex objects are provided as inputs to the trained network without additional training. As shown in Table 5.1, network (iii) achieves the lowest RE (3.57%) for these generalized data, which shows the improved generalization capability of the proposed network. As these hand-drawn shapes are quite different from the regular shapes in the training set, the RE obtained for the generalized data is higher than that for the regular data but still satisfactory. The imaging results (Cases 7-9) are compared in Figure 5.5. The predicted B-scans match well with the ones obtained by the FDTD solver; the differences between sets of B-scans are very small. These results demonstrate the excellent generalization capability of the proposed method for random convex objects.

2) New Scenarios

To verify the effectiveness of the proposed transfer learning algorithm, 528 sets of data (480 for training and 48 for testing) are generated for two new scenarios: (1) the random distribution of the subsurface heterogeneous background environment is replaced with a distribution different than those in the pre-training set, and (2) a commercial MALA 1.2 GHz antenna model is replaced with the previous dipole-probe configuration as the transceiver for GPR scanning. The REs of the testing results using three models are compared in Table 5.2. In particular, models (i), (ii), and (iii) represent the pre-trained model as described in Section 5.3.1, the model re-trained using new training data from scratch, and the model fine-tuned using transfer learning, respectively. As shown, the prediction accuracy of model (i) is the worst, as the testing data for new scenarios are vastly different from the pre-training data, especially for the cases with the totally new antenna model. For model (ii) which performs the training from scratch using a small amount of data, the REs are still large. This is because the network is underfitting and is far from being well-trained with a very limited set of training data. However, model (iii) leveraging the transfer learning technique to effectively transfer the well-learned non-linear mapping relationship in the pre-training stage to new tasks, can accurately predict the B-scans for these new scenarios.

The imaging results for the new scenarios are shown in Figure 5.6. The B-scans predicted using the model (iii) match well with the ones obtained by the FDTD solver. The absolute differences between pixel values are quite small. The quantitative and qualitative analyses demonstrate that the transfer learning-applied network shows a significantly enhanced generalization capability for the new scenarios.

Table 5.2: Evaluation metrics comparison for the new soil distribution and antenna model.

New Scenarios	Model (i)	Model (ii)	Model (iii)
(1) New Distribution	6.88%	4.07%	**1.20%**
(2) New Antenna Model	126.10%	12.94%	**3.91%**

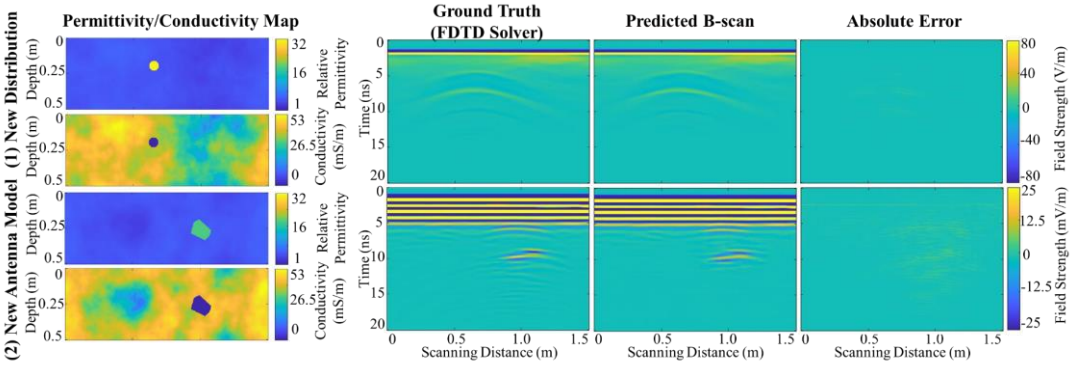


Figure 5.6: The imaging results for the new soil distribution and antenna model.

3) Different Time Windows and Lengths of Survey Lines

To further verify the applicability of the proposed method for different time windows and lengths of survey lines, we generated two small sets of new data using the physics-based solver for the following cases: (1) The time window T, the depth of the subsurface scenario D, and the length of survey line L (the scanning distance) are reduced to 15 ns, 0.375 m, and 1.125 m, respectively, the dimension of the input subsurface permittivity/conductivity map $H \times W$ becomes 150×450, the dimension of the corresponding output B-scan becomes 192×224. (2) The T, D, and L are increased to 25 ns, 0.625 m, and 1.875 m, respectively, the $H \times W$ becomes 250×750, and the dimension of the corresponding output B-scan becomes 288×320. It should be noted that in the pre-training set, the T, D, L, and $H \times W$ are 20 ns, 0.5 m, 1.5 m, and 200×600, respectively. The subsurface heterogeneous soil distributions of the new-dimension domains are also different. Other parameter settings remain unchanged. In each dataset, 480 sets of permittivity map, conductivity map, and the corresponding B-scan with new dimensions and time windows are used to fine-tune the pre-trained network and 48 sets are used to test the network performance. In the fine-tuning process, the learning rate is set as low as 2.5×10^{-5} to avoid over-fitting.

The REs of the testing data with new dimensions and time windows are shown in Table 5.3. The low REs demonstrate that the network can accurately predict the B-scans for these new cases through transfer learning. The imaging results are shown in Figure 5.7. The predicted B-scans match well with the ones obtained by the physics-based solver. The absolute differences between pixel values are quite small. These

quantitative and qualitative analyses show the generalization capability of the transfer learning-applied network to the scenarios with new dimensions and time windows.

Table 5.3: REs of testing data with new dimensions and time windows.

Data Type	Parameters				RE
	D (m)	L (m)	T (ns)	$H \times W$	
Pre-Training	0.5	1.5	20	200×600	1.28%
(1)	0.375	1.125	15	150×450	1.03%
(2)	0.625	1.875	25	250×750	1.17%

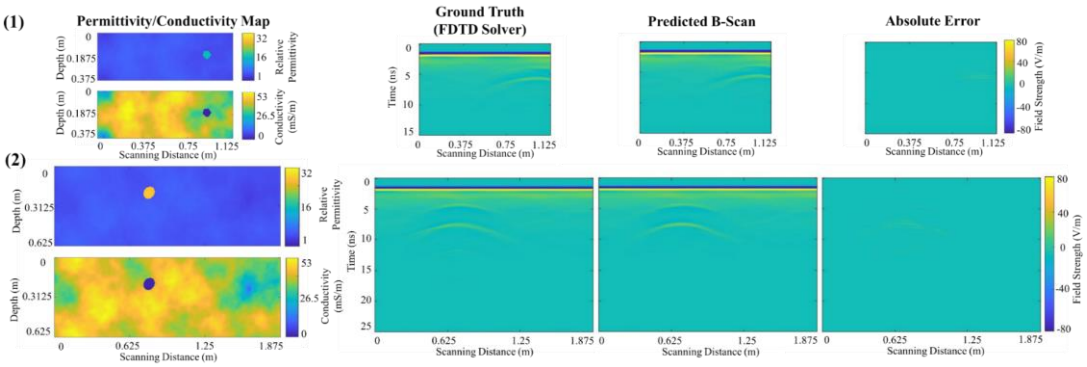


Figure 5.7: Imaging results for subsurface scenarios with new dimensions and time windows.

4) Concave Objects

We further test the generalization capability of the proposed solver for concave objects including slightly concave objects and severely concave objects. The test scenarios and imaging results are presented in Figure 5.8. As shown in Figure 5.8(a), two sets of permittivity and conductivity maps with hand-drawn slightly concave objects buried in heterogeneous environments are provided as inputs to the pre-trained network without additional training. For these slightly concave objects, the main reflection patterns predicted by the proposed network are close to those in the ground truth images obtained by the FDTD solver, but some multiple reflections inside the objects cannot be predicted precisely. We also calculate the MRE for these slightly concave cases to quantitatively evaluate the performance. The average MRE is 5.33%, which is much worse than that for the regular convex-object scenarios

(1.28%) but is still acceptable.

Furthermore, to test the performance of the solver on severely concave objects, as shown in Figure 5.8(b), two sets of permittivity and conductivity maps with hand-drawn severely concave objects buried in heterogeneous environments are provided as inputs to the pre-trained network without additional training. For these severely concave objects with vastly different shapes compared to the convex objects in the pre-training set, the sizes and positions of the main reflection patterns predicted by the proposed network are accurate, but the values and shapes are far from the ones obtained by the FDTD solver, especially the multiple reflections inside the severely concave objects cannot be well predicted by the network. The average MRE of these severely concave cases is as high as 7.50%.

These test results demonstrate that the proposed solver has the potential to be generalized to slightly concave cases even though the pre-training set only includes convex-object cases. However, for severely-concave-object scenarios that contain obvious multiple reflections inside the object, the network cannot accurately predict their corresponding B-scans.

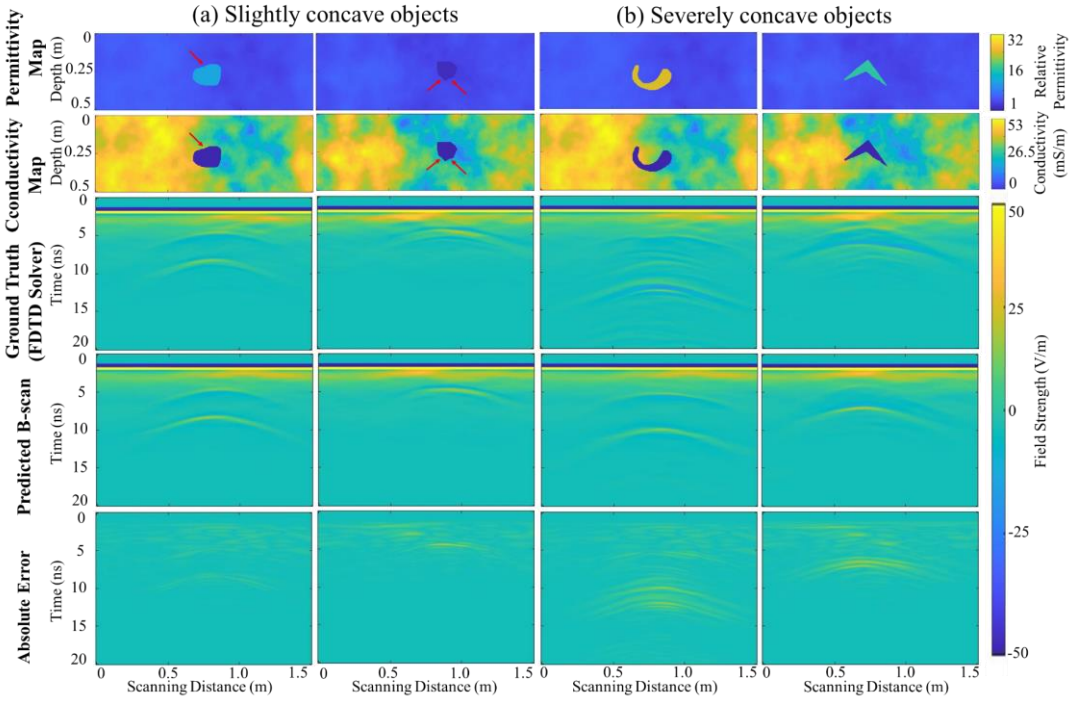


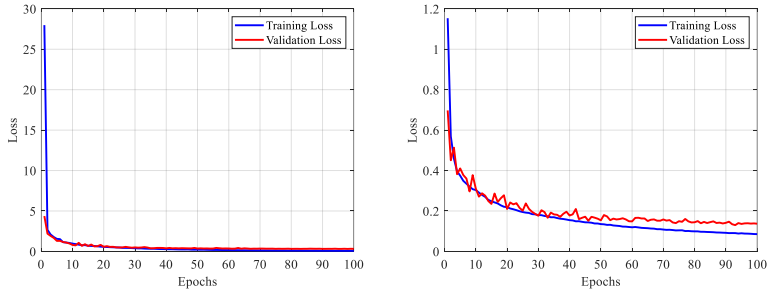
Figure 5.8: Test results for concave-object scenarios including (a) slightly concave objects and (b) severely concave objects.

5.3.4 Comparative Study on Different Loss Functions

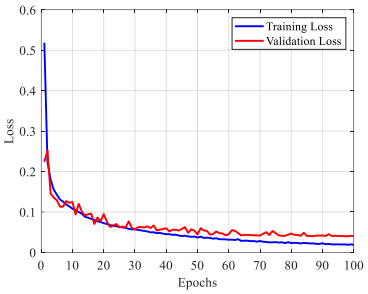
The purpose of this work is to solve a regression problem that involves predicting real-valued quantities (the field strength values in B-scans). For a deep learning-based regression problem, there are three widely used loss functions for training the network: 1) MSE, 2) MAE, and 3) Pseudo-Huber loss (Huber) [134]. MSE is a quadratic scoring method, and the penalty is proportional to the square of the error. It gives a relatively higher weight (penalty) to large errors and smoothens the gradient for smaller errors. MAE is a linear scoring method that weights all errors equally when calculating the mean. It is more suitable when the data distribution includes outliers. Huber is a loss function that balances the MSE and MAE using a hyperparameter δ. If the absolute error is over δ, it changes the quadratic scoring to the linear one.

We first compare the training efficiency of the three loss functions. Figure 5.9 show the training and validation loss curves using MSE, MAE, and Huber when δ is set as 0.5, 1.0, and 2.0, respectively. From the validation loss curves, it can be observed that the MSE achieves the fastest convergence rate. This characteristic of MSE improves the efficiency of network training using a large diverse set of data in the pre-training stage.

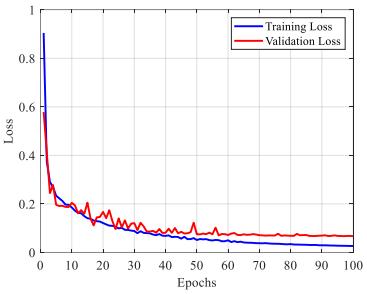
Then we compare the performance of the network trained with different loss functions as listed in Table 5.4. The testing cases include the regular testing data and the generalized testing data with hand-drawn objects. The REs using MSE as the loss function are the lowest, demonstrating the superiority of MSE in training the network to accurately predict B-scans from provided permittivity and conductivity maps.



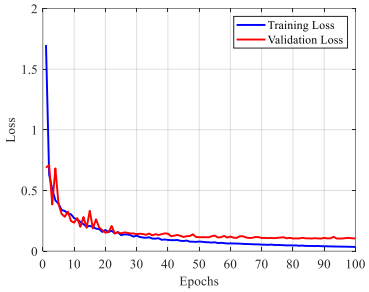
(a) MSE (b) MAE



(c) Huber ($\delta = 0.5$)



(d) Huber ($\delta = 1.0$)



(e) Huber ($\delta = 2.0$)

Figure 5.9: Training and validation loss curves using different loss functions. MSE and MAE represent the mean square error and the mean absolute error, respectively. Huber is a loss function that balances the MSE and MAE using the hyperparameter δ.

Table 5.4: Evaluation metrics comparison using different loss functions.

Loss Function	REs	
	Regular Testing Data	Generalized Testing Data
MSE	**1.28%**	**3.57%**
MAE	2.25%	4.74%
Huber ($\delta = 0.5$)	2.41%	4.55%
Huber ($\delta = 1.0$)	2.11%	4.77%
Huber ($\delta = 2.0$)	2.39%	4.69%

5.4 Conclusion

In this work, a deep learning-based 2D GPR forward solver was proposed to obtain the B-scans for subsurface convex objects. In particular, a novel bimodal encoder-

decoder neural network with the adaptive feature fusion module was designed to learn the non-linear mapping relationship between the subsurface permittivity and conductivity maps and the corresponding B-scan. Transfer learning was introduced to enhance the network's generalization capability. Given the permittivity and conductivity maps of a random convex object, the network can predict the corresponding B-scan accurately and efficiently. The computation time required by the proposed network to predict one B-scan is 22,500x less than that required by the classical FDTD solver. With its high accuracy and efficiency, the proposed network could be highly beneficial for GPR applications requiring forward solvers, such as full-wave inversion and data-driven GPR techniques. It should be noted that the performance on concave-object scenarios is degraded compared with the convex-object scenarios, which are the focus of this study. Our future work will investigate advanced deep learning solvers for non-convex scenarios.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, two deep learning-based schemes are proposed to solve GPR inverse problems under heterogeneous soil conditions, and one fast GPR forward solver is proposed for accelerating the dataset generation in the deep learning-based inversion algorithms that require large sets of GPR data for training the DNNs. We conclude our works as follows.

First, for the 2D GPR data inversion under heterogeneous soil conditions, a two-stage network DMRF-UNet is designed to reconstruct the subsurface 2D permittivity distribution from the GPR B-scan. The first stage employs a U-shape DNN to extract the object signatures and remove the noise and clutter in the noisy B-scan. The second stage takes the concatenated denoised image and noisy image as a hybrid input and reconstructs the permittivity distributions of subsurface objects. An end-to-end training method that combines the losses of two stages is used to avoid information loss. The results demonstrate that the proposed network allows a robust and accurate reconstruction of the 2D permittivity distributions of subsurface objects and outperforms existing techniques, especially for complex scenarios with multiple objects. This work addresses the challenges of GPR inverse problems under heterogeneous soil conditions using deep learning techniques. It demonstrates the potential of deep learning-based schemes for real-world inversion applications.

Second, for the 3D GPR data inversion that considers more object information in 3D domain, a two-stage scheme called 3DInvNet is proposed to rapidly denoise GPR C-scans and reconstruct 3D subsurface permittivity maps. The first stage Denoiser extracts the informative features of subsurface objects and suppresses the environmental noise in the GPR C-scans. The second stage Inverter builds the relationship between the discriminative features of the denoised C-scans with the corresponding subsurface scenarios. The results of tests on both the numerical and

real measurement data demonstrate that the proposed method can reconstruct 3D permittivity maps with high accuracy and efficiency. This work is the first that implements a 3D deep learning technique for 3D subsurface permittivity map reconstruction. It provides a novel and efficient method to solve the GPR 3D inverse problem and alleviate the computational burden of 3D reconstruction.

In addition, to reduce the computational costs of the training stage of the deep learning-based GPR inversion that requires a large and diverse dataset, a fast GPR forward solver is proposed to generate GPR B-scans when the subsurface permittivity and conductivity maps are provided. A novel bimodal encoder-decoder neural network with the adaptive feature fusion module is designed to learn the non-linear mapping relationship between the subsurface permittivity and conductivity maps and the corresponding B-scan. Furthermore, transfer learning is introduced to enhance the network's generalization capability. The results show that the network can predict the B-scan accurately and efficiently for any random convex object in near-real-time. With its high accuracy and efficiency, the proposed solver could be highly useful for alleviating the computational burden of iterative FWI algorithms, data-driven GPR inversion techniques, and other GPR applications that requires repetitive GPR forward modeling.

Note: The implemented codes of the above three works can be found at https://github.com/Qiqi-Dai/DMRF-UNet, https://github.com/Qiqi-Dai/3DInvNet, and https://github.com/Qiqi-Dai/GPRForwardSolver, respectively.

6.2 Future Work

Four recommendations for future work are proposed in this section.

First, for reconstructing more real-world scenarios including metallic targets, a small new dataset with buried metallic targets is required to fine-tune the pre-trained networks. As the permittivity value of a metallic target is infinite, we can set it to a much larger value to separate it from other objects. In the future, such dataset will be collected and real experiments for more complicated scenarios will be conducted to further verify the proposed schemes.

Second, to form a precise mapping between the GPR data and subsurface information via a deep learning model in GPR inverse problems, a large labeled GPR dataset is

required for training and testing the DNNs. Nevertheless, it is challenging to collect massive GPR data in the real world, as the repetitive measurements require excessive time, resources, and labour. On the other hand, the training and testing of the deep learning algorithms with the synthetic data obtained via simulations are often misleading due to the significant differences between synthetic and measurement data. To solve this issue, our future work will develop advanced generative adversarial networks to translate synthetic GPR B-scans for known subsurface scenarios into the corresponding measured ones. The generated measurement data will be used for data augmentation and applied to deep learning-based GPR inversion techniques, such as GPR image classification, object detection, and subsurface structural image reconstruction.

Third, denoising the measured GPR data plays an important role in GPR inverse problems. Existing supervised deep learning-based denoising methods require clean noise-free data as labels to train the networks. However, in real-field applications, obtaining the noise-free GPR data is highly challenging. To address this problem, our future work will focus on designing an unsupervised leaning-based method to denoise the measured data. Generative adversarial networks will be used to learn the features of noisy patterns obtained in the real field. Using the trained network, the noise of the measured data will be automatically removed, and the object signatures will be extracted. As unsupervised learning requires no noise-free GPR data as labels for training the network, the applicability of the denoising technique to new real fields will be enhanced.

Fourth, when the background dielectric parameters of the soil medium are not known, it is difficult for the GPR inversion network to image the subsurface structure. One potential solution is to develop a prior deep learning model to predict the background dielectric parameters. In particular, we can develop a DNN for predicting the average permittivity and the average conductivity of the background. The network structure can be built based on the mask-guided multi-polarimetric integration neural network (MMI-Net) proposed in our previous work [135], which automatically and simultaneously estimates five object-related parameters in the fixed soil environment. Two more neurons can be added to the final layer to output the average permittivity and the average conductivity of the soil. A more diverse dataset that considers different soil environments can be used to train and test the expanded network.

Another possible solution is to combine physics-based algorithms that aim to estimate the dielectric parameters of the subsurface soil. After that, the subsequent inversion algorithms will take the known background dielectric parameters as labels to train the network and reconstruct the subsurface structural image. The combination with physics-based method will improve the independence on large datasets of data-driven techniques, such as the deep learning-based schemes, and make them applicable to any unseen fields with new background dielectric parameters, which will be further investigated in our future work.

Author's Publications

Journal Articles:

1. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "DMRF-UNet: A two-stage deep learning scheme for GPR data inversion under heterogeneous soil conditions," *IEEE Transactions on Antennas and Propagation*, vol. 70, no. 8, pp. 6313-6328, 2022.
2. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "3DInvNet: A Deep Learning-Based 3D Ground-Penetrating Radar Data Inversion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1-16, 2023.
3. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "A deep learning-based GPR forward solver for predicting B-scans of subsurface objects," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022.
4. Hai-Han Sun, Yee Hui Lee, **Qiqi Dai**, Chongyi Li, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "Estimating parameters of the tree root in heterogeneous soil environments via mask-guided multi-polarimetric integration neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-16, 2022.

Conference Papers:

1. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "S2M-Net: A deep learning-based scheme for GPR image translation from simulation to measurement via a conditional generative adversarial network," in *Proceedings of IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Denver, Colorado, USA, 2022.
2. Jiwei Qian, Yee Hui Lee, **Qiqi Dai**, Kaixuan Cheng, Liansheng He, Daryl Lee, and Abdulkadir C. Yucel, "Rapid health assessment of trees via deep learning-augmented radar," in *Proceedings of IEEE International Symposium on*

Antennas and Propagation and USNC-URSI Radio Science Meeting, Denver, Colorado, USA, 2022.

3. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "A deep learning scheme for rapidly reconstructing 3D permittivity maps from GPR C-scans," in *Proceedings of IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Singapore, 2021.
4. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "A two-stage deep neural network for ground-penetrating radar data inversion under heterogeneous soil conditions," in *Proceedings of IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Singapore, 2021.
5. **Qiqi Dai**, Yee Hui Lee, Hai-Han Sun, Genevieve Ow, Mohamed Lokman Mohd Yusof, and Abdulkadir C. Yucel, "A fast 2D GPR forward solver for convex objects based on a deep learning technique," in *Proceedings of IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Singapore, 2021.
6. **Qiqi Dai**, Bihan Wen, Genevieve Ow, Mohamed Lokman Mohd Yusof, Yee Hui Lee, and Abdulkadir C. Yucel, "A deep learning-based methodology for rapidly detecting the defects inside tree trunks via GPR," in *Proceedings of IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Montreal, Canada, 2020.
7. **Qiqi Dai**, Yee Hui Lee, Mohamed Lokman Mohd Yusof, Daryl Lee, and Abdulkadir C. Yucel, "Learning from noise: An unsupervised GPR data denoising scheme based on generative adversarial networks," accepted, *IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Portland, Oregon, USA, 2023.
8. **Qiqi Dai**, Yee Hui Lee, Jiwei Qian, Mohamed Lokman Mohd Yusof, Daryl Lee, and Abdulkadir C. Yucel, "A signal processing algorithms-assisted deep learning scheme for ground-penetrating radar imaging," accepted, *IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, Portland, Oregon, USA, 2023.

References

[1] H. M. Jol, *Ground Penetrating Radar: Theory and Applications*. Amsterdam, The Netherlands: Elsevier, 2008.

[2] R. González-Drigo, V. Pérez-Gracia, D. Di Capua, and L. G. Pujades, "GPR survey applied to Modernista buildings in Barcelona: The cultural heritage of the College of Industrial Engineering," *Journal of Cultural Heritage*, vol. 9, no. 2, pp.196-202, 2008.

[3] V. Barrile and R. Pucinotti, "Application of radar technology to reinforced concrete structures: A case study," N*DT & e International*, vol. 38, no. 7, pp.596-604, 2005.

[4] N. Barraca, M. Almeida, H. Varum, F. Almeida, and M. S. Matias, "A case study of the use of GPR for rehabilitation of a classified Art Deco building: The InovaDomus house," *Journal of Applied Geophysics*, vol. 127, pp.1-13, 2016.

[5] M. Solla, H. González-Jorge, H. Lorenzo, and P. Arias, "Uncertainty evaluation of the 1 GHz GPR antenna for the estimation of concrete asphalt thickness," *Measurement*, vol. 46, no. 9, pp.3032-3040, 2013.

[6] Z. Leng and I. L. Al-Qadi, "An innovative method for measuring pavement dielectric constant using the extended CMP method with two air-coupled GPR systems," *NDT & e International*, vol. 66, pp.90-98, 2014.

[7] F. Zhang, X. Xie, and H. Huang, "Application of ground penetrating radar in grouting evaluation for shield tunnel construction," *Tunnelling and Underground Space Technology*, vol. 25, no. 2, pp.99-107, 2010.

[8] A. Lalagüe, M. A. Lebens, I. Hoff, and E. Grøv, "Detection of rockfall on a tunnel concrete lining with ground-penetrating radar (GPR)," *Rock Mechanics and Rock Engineering*, vol. 49, pp.2811-2823, 2016.

[9] R. Gerber, C. Salat, A. Junge, and P. Felix-Henningsen, "GPR-based detection of Pleistocene periglacial slope deposits at a shallow-depth test site," *Geoderma*, vol. 139, pp.346-356, 2007.

[10] D. Gómez-Ortiz and T. Martín-Crespo, "Assessing the risk of subsidence of a sinkhole collapse using ground penetrating radar and electrical resistivity tomography," *Engineering Geology*, vol. 149, pp.1-12, 2012.

[11] J. Lester and L. E. Bernold, "Innovative process to characterize buried utilities using ground penetrating radar," *Automation in Construction*, vol. 16, no. 4, pp.546-555, 2007.

[12] U. S. Khan, W. Al-Nuaimy, and E. F. Abd El-Samie, "Detection of landmines and underground utilities from acoustic and GPR images with a cepstral approach," *Journal of Visual Communication and Image Representation*, vol. 21, no. 7, pp.731-740, 2010.

[13] Y. Jeng and C. S. Chen, "Subsurface GPR imaging of a potential collapse area in urban environments," *Engineering Geology*, vol. 147, pp.57-67, 2012.

[14] H. Qin, X. Xie, J. A. Vrugt, K. Zeng, and G. Hong, "Underground structure defect detection and reconstruction using crosshole GPR and Bayesian waveform inversion," *Automation in Construction*, vol. 68, pp. 156-169, 2016.

[15] D. Feng, X. Wang, and B. Zhang, "Improving reconstruction of tunnel lining defects from ground-penetrating radar profiles by multi-scale inversion and

bi-parametric full-waveform inversion," *Advanced Engineering Informatics*, vol. 41, p. 100931, 2019.

[16] Z. Tong, J. Gao, and D. Yuan, "Advances of deep learning applications in ground-penetrating radar: A survey," *Construction Building Materials*, vol. 258, p. 120371, 2020.

[17] M. Küçükdemirci and A. Sarris, "GPR data processing and interpretation based on artificial intelligence approaches: Future perspectives for archaeological prospection, " *Remote Sensing*, vol. 14, no. 14, p. 3377, 2022.

[18] J. K. Alvarez and S. Kodagoda, "Application of deep learning image-to-image transformation networks to GPR radargrams for sub-surface imaging in infrastructure monitoring," in *Proceedings of IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018: IEEE, pp. 611-616.

[19] Y. Pang, J. Lin, T. Qin, and Z. Chen, "Image-to-image translation: Methods and applications," *IEEE Transactions on Multimedia*, vol. 24, pp. 3859-3881, 2022.

[20] R. Yelf, "Where is true time zero?, " in *Proceedings of the Tenth International Conference on Grounds Penetrating Radar*, 2004: IEEE, vol. 1, pp. 279-282.

[21] A. Benedetto, F. Tosti, L. B. Ciampoli, and F. D'Amico, "An overview of ground-penetrating radar signal processing techniques for road inspections," *Signal Processing*, vol. 132, pp. 201-209, 2017.

[22] J. H. Kim, S. J. Cho, and M. J. Yi, "Removal of ringing noise in GPR data by signal processing," *Geosciences Journal*, vol. 11, pp. 75-81, 2007.

[23] B. Cagnoli and T. J. Ulrych, "Singular value decomposition and wavy reflections in ground-penetrating radar images of base surge deposits," *Journal of Applied Geophysics*, vol. 48, no. 3, pp. 175-182, 2001.

[24] C. Özdemir, Ş. Demirci, E. Yiğit, and B. Yilmaz, "A review on migration methods in B-scan ground penetrating radar imaging," *Mathematical Problems in Engineering*, vol. 2014, pp. 1-16, 2014.

[25] I. Catapano, A. Randazzo, E. Slob, and R. Solimene, "GPR imaging via qualitative and quantitative approaches," in *Civil Engineering Applications of Ground Penetrating Radar*, A. Benedetto and L. Pajewski, Eds. Cham, Switzerland: Springer, 2015, pp. 239-280.

[26] E. Baysal, D. D. Kosloff, and J. W. C. Sherwood, "Reverse time migration," *Geophysics*, vol. 48, no. 11, pp. 1514-1524, 1983.

[27] Y. Lyu, H. Wang, and J. Gong, "GPR Detection of tunnel lining cavities and reverse-time migration imaging," *Applied Geophysics*, vol. 17, no. 1, pp. 1-7, 2020.

[28] M. L. Moran, R. J. Greenfield, S. A. Arcone, and A. J. Delaney, "Multidimensional GPR array processing using Kirchhoff migration," *Journal of Applied Geophysics*, vol. 43, no. 2-4, pp. 281-295, 2000.

[29] X. Xu, E. L. Miller, and C. M. Rappaport, "Minimum entropy regularization in frequency-wavenumber migration to localize subsurface objects," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 8, pp. 1804-1812, 2003.

[30] L. Crocco, G. Prisco, F. Soldovieri, and N. J. Cassidy, "Early-stage leaking pipes GPR monitoring via microwave tomographic inversion," *Journal of Applied Geophysics*, vol. 67, no. 4, pp.270-277, 2009.

[31] R. Persico and F. Soldovieri, "A microwave tomography approach for a differential configuration in GPR prospecting," *IEEE Transactions on Antennas and Propagation*, vol. 54, no. 11, pp. 3541-3548, 2006.

[32] M. Salucci, L. Poli, N. Anselmi, and A. Massa, "Multifrequency particle swarm optimization for enhanced multiresolution GPR microwave imaging," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 3, pp. 1305-1317, 2016.

[33] I. Catapano, G. Gennarelli, G. Ludeno, and F. Soldovieri, "Applying ground-penetrating radar and microwave tomography data processing in cultural heritage: State of the art and future trends," *IEEE Signal Processing Magazine*, vol. 36, no. 4, pp.53-61, 2019.

[34] G. Meles, S. Greenhalgh, J. Van der Kruk, A. Green, and H. Maurer, "Taming the non-linearity problem in GPR full-waveform inversion for high contrast media," *Journal of Applied Geophysics*, vol. 78, pp.31-43, 2012.

[35] D. Feng, C. Cao, and X. Wang, "Multiscale full-waveform dual-parameter inversion based on total variation regularization to on-ground GPR data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 11, pp. 9450-9465, 2019.

[36] F. Lavoué, R. Brossier, L. Métivier, S. Garambois, and J. Virieux, "Two-dimensional permittivity and conductivity imaging by full waveform inversion of multioffset GPR data: A frequency-domain quasi-Newton approach," *Geophysical Journal International*, vol. 197, no. 1, pp. 248-268, 2014.

[37] X. Wang, S. Sun, J. Wang, A. Yarovoy, B. Neducza, and G. Manacorda, "Real GPR signal processing for target recognition with circular array antenna," in *Proceedings of URSI International Symposium on Electromagnetic Theory*, 2016: IEEE. pp. 818-821.

[38] M. Pereira, Y. Zhang, D. Orfeo, D. Burns, D. Huston, and T. Xia, "3D tomographic image reconstruction for multistatic ground penetrating radar," in *Proceedings of IEEE Radar Conference (RadarConf)*, 2019: IEEE, pp. 1-6.

[39] M. Pereira, D. Burns, D. Orfeo, Y. Zhang, L. Jiao, D. Huston and T. Xia, "3-D multistatic ground penetrating radar imaging for augmented reality visualization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 8, pp. 5666-5675, 2020.

[40] L. Zou, Y. Wang, I. Giannakis, F. Tosti, A. M. Alani, and M. Sato, "Mapping and assessment of tree roots using ground penetrating radar with low-cost GPS," *Remote Sensing*, vol. 12, no. 8, p. 1300, 2020.

[41] M. Wang, H. Wang, and H. Liu, "3D pre-stack reverse time migration of ground penetrating radar for subsurface imaging," in P*roceeding of International Conference on Ground Penetrating Radar (GPR)*, 2018: IEEE, pp. 1-4.

[42] W. Zhu, Q. Huang, L. Liu, and B. Ma, "Three-dimensional reverse time migration of ground-penetrating radar signals," *Pure and Applied Geophysics*, vol. 177, no. 2, pp. 853-865, 2020.

[43] F. Watson, "Towards 3D full-wave inversion for GPR," in *Proceeding of IEEE Radar Conference (RadarConf)*, 2016: IEEE, pp. 1-6.

[44] X. Wang and D. Feng, "Multiparameter full-waveform inversion of 3-D on-ground GPR with a modified total variation regularization scheme," *IEEE Geoscience and Remote Sensing Letters*, vol. 18, no. 3, pp. 466-470, 2021.

[45] L. E. Besaw and P. J. Stimac, "Deep convolutional neural networks for classifying GPR B-scans," in *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XX*, 2015, vol. 9454: International Society for Optics and Photonics, p. 945413.

[46] J. Zheng, X. Teng, J. Liu, and X. Qiao, "Convolutional neural networks for water content classification and prediction with ground penetrating radar," *IEEE Access*, vol. 7, pp.185385-185392, 2019.

[47] U. Ozkaya, F. Melgani, M. B. Bejiga, L. Seyfi, and M. Donelli, "GPR B scan image analysis with deep learning methods," *Measurement*, vol. 165, p.107770, 2020.

[48] N. Kim, S. Kim, Y. K. An, and J. J. Lee, "A novel 3D GPR image arrangement for deep learning-based underground object classification," *International Journal of Pavement Engineering*, vol. 22, no. 6, pp. 740-751, 2021.

[49] M. T. Pham, and S. Lefèvre, "Buried object detection from B-scan ground penetrating radar data using Faster-RCNN," in *Proceedings of IEEE International Geoscience and Remote Sensing Symposium*, 2018: IEEE, pp. 6804-6807.

[50] F. Hou, W. Lei, S. Li, and J. Xi, "Deep learning-based subsurface target detection from GPR scans," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 8161-8171, 2021.

[51] Z. Liu, X. Gu, W. Wu, X. Zou, Q. Dong, and L. Wang, "GPR-based detection of internal cracks in asphalt pavement: A combination method of DeepAugment data and object detection," *Measurement*, vol. 197, p.111281, 2022.

[52] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017.

[53] Z. Huang, L. Huang, Y. Gong, C. Huang, and X. Wang, "Mask Scoring R-CNN," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6409-6418.

[54] J. Redmon and A. Farhadi, "YOLOV3: An incremental improvement," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017: IEEE, pp. 1-6.

[55] H. -H. Sun, Y. H. Lee, C. Li, G. Ow, M. L. M. Yusof, and A. C. Yucel, "The orientation estimation of elongated underground objects via multi-polarization aggregation and selection neural network," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2021.

[56] H. Wang, S. Ouyang, Q. Liu, K. Liao, and L. Zhou, "Deep-learning-based method for estimating permittivity of ground-penetrating radar targets," *Remote Sensing*, vol. 14, no. 17, p. 4293, 2022.

[57] L. Xie, Q. Zhao, C. Ma, B. Liao, and J. Huo, "Ü-Net: Deep-learning schemes for ground penetrating radar data inversion," *Journal of Environmental and Engineering Geophysics*, vol. 25, no. 2, pp. 287-292, 2020.

[58] B. Liu, Y. Ren, H. Liu, H. Xu, Z. Wang, A. G. Cohn, and P. Jiang, "GPRInvNet: Deep learning-based ground-penetrating radar data inversion for tunnel linings," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 10, pp. 8305-8325, 2021.

[59] Y. Ji, F. Zhang, J. Wang, Z. Wang, P. Jiang, H. Liu, and Q. Sui, "Deep neural network-based permittivity inversions for ground penetrating radar data," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 8172-8183, 2021.

[60] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481-2495, 2017.

[61] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015: Springer, pp. 234-241.

[62] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017: IEEE, pp. 5967-5976.

[63] D. Goodman, Y. Nishimura, and K. Tobita, "GPRSIM forward modeling software and time slices in ground penetrating radar surveys," in *Proceedings of the Fifth International Conferention on Ground Penetrating Radar (GPR)*, 1994, pp. 31-43.

[64] CST STUDIO SUITE™, CST AG, Germany, www.cst.com.

[65] 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, 2016.

[66] C. Warren, A. Giannopoulos, A. Gray, I. Giannakis, A. Patterson, L. Wetter, and A. Hamrah, "A CUDA-based GPU engine for gprMax: Open source FDTD electromagnetic simulation software," *Computer Physics Communications*, vol. 237, pp. 208-218, 2019.

[67] M. Zych, "Ground penetrating radar simulations of non-homogeneous soil with CST Studio Suite," *Przegląd Elektrotechniczny*, vol. 89, no. 7, pp.182-185, 2013.

[68] A. Joret, M. F. L. Abdullah, and M. S. Sulong, "Simulation of GPR system design using CST microwave and Matlab," *Yanbu Journal of Engineering and Science*, vol. 14, no. 1, pp.49-57, 2021.

[69] K. S. Kunz and R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*. CRC Press, 1993.

[70] I. Giannakis, A. Giannopoulos, and C. Warren, "A realistic FDTD numerical modeling framework of ground penetrating radar for landmine detection," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 1, pp. 37-51, 2015.

[71] I. Giannakis and A. Giannopoulos, "A novel piecewise linear recursive convolution approach for dispersive media using the finite-difference time-domain method," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 5, pp. 2669-2678, 2014.

[72] N. R. Peplinski, F. T. Ulaby, and M. C. Dobson, "Dielectric properties of soils in the 0.3-1.3-GHz range," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 3, pp. 803-807, 1995.

[73] D. Goodman, "Ground-penetrating radar simulation in engineering and archaeology," *Geophysics*, vol. 59, no. 2, pp. 224-232, 1994.

[74] H. W. Chen and T. M. Huang, "Finite-difference time-domain simulation of GPR data, " *Journal of Applied Geophysics*, vol. 40, no. 1-3, pp. 139-163, 1998.

[75] F. L. Teixeira, W. C. Chew, M. Straka, M. L. Oristaglio, and T. Wang, "Finite-difference time-domain simulation of ground penetrating radar on dispersive, inhomogeneous, and conductive soils," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 36, no. 6, pp. 1928-1937, 1998.

[76] I. Van den Bosch, S. Lambot, M. Acheroy, I. Huynen, and P. Druyts, "Accurate and efficient modeling of monostatic GPR signal of dielectric

targets buried in stratified media," *Journal of Electromagnetic Waves and Applications*, vol. 20, no. 3, pp. 283-290, 2006.

[77] H. Liu, B. Xing, H. Wang, J. Cui, and B. F. Spencer, "Simulation of ground penetrating radar on dispersive media by a finite element time domain algorithm," *Journal of Applied Geophysics*, vol. 170, p. 103821, 2019.

[78] H. Wang, Y. Lv, M. Wang, J, Gong, and Z. Zhang, "A perfectly matched layer for second order electromagnetic wave simulation of GPR by finite element time domain method, " *Chinese Journal of Geophysics*, vol. 62, no. 5, pp. 1929-1941, 2019.

[79] A. Massa, D. Marcantonio, X. Chen, M. Li, and M. Salucci, "DNNs as applied to electromagnetics, antennas, and propagation—A review, " *IEEE Antennas and Wireless Propagation Letters*, vol. 18, no. 11, pp. 2225-2229, 2019.

[80] T. M. Hansen, and K. S. Cordua, "Efficient Monte Carlo sampling of inverse problems using a neural network-based forward—Applied to GPR crosshole traveltime inversion," *Geophysical Journal International*, vol. 211, no. 3, pp. 1524-1533, 2017.

[81] I. Giannakis, A. Giannopoulos, and C. Warren, "A machine learning approach for simulating ground penetrating radar," in *Proceedings of International Conference on Ground Penetrating Radar (GPR)*, 2018: IEEE, pp. 1-4.

[82] I. Giannakis, A. Giannopoulos, and C. Warren, "A machine learning-based fast-forward solver for ground penetrating radar with application to full-waveform inversion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 7, pp. 4417-4426, 2019.

[83] C. A. Balanis, *Advanced Engineering Electromagnetics*. John Wiley & Sons, 2012.

[84] G. R. Naik and W. Wang, "Blind source separation: Advances in theory, algorithms and applications", *Signals and Communication Technology*. Berlin, Germany: Springer, 2014.

[85] D. Kumlu and I. Erer, "Clutter removal techniques in ground penetrating radar for landmine detection: A survey," in *Operations Research for Military Organizations*. Hershey, PA, USA: IGI Global, 2019.

[86] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017: IEEE, pp. 4700-4708.

[87] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015: IEEE, pp. 1-9.

[88] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016: IEEE, pp. 2818-2826.

[89] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in Proceddings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017: AAAI, p. 12.

[90] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Proceedings of Advances in Neural Information Processing Systems*, 2016, pp. 4898-4906.

[91] A. Araujo, W. Norris, and J. Sim, "Computing receptive fields of convolutional neural networks," *Distill*, vol. 4, no. 11, p. 151, 2019.

[92] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 265-283.

[93] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2014.

[94] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, 2004.

[95] S. Jazayeri, S. Kruse, I. Hasan, and N. Yazdani, "Reinforced concrete mapping using full-waveform inversion of GPR data," *Construction and Building Materials*, vol. 229, p. 117102, 2019.

[96] I. Giannakis, A. Giannopoulos, and C. Warren, "A machine learning-based fast-forward solver for ground penetrating radar with application to full-waveform inversion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 7, pp. 4417-4426, 2019.

[97] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[98] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Proceedings of International Conference on Artificial Neural Networks*, 2018: Springer, pp. 270-279.

[99] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2323, 1998.

[100] R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen, "Ablation studies in artificial neural networks," *arXiv:1901.08644*, 2019.

[101] H. Brunzell, "Detection of shallowly buried objects using impulse radar," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 2, pp. 875–886, 1999.

[102] A. M. Zoubir, I. J. Chant, C. L. Brown, B. Barkat, and C. Abeynayake, "Signal processing techniques for landmine detection using impulse ground penetrating radar," *IEEE Sensor Journal*, vol. 2, no. 1, pp. 41–51, 2002.

[103] A. Aboudourib, M. Serhir, and D. Lesselier, "A processing framework for tree-root reconstruction using ground-penetrating radar under heterogeneous soil conditions, " *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 1, pp. 208-219, 2020.

[104] F. Abujarad, G. Nadim, and A. Omar, "Clutter reduction and detection of landmine objects in ground penetrating radar data using singular value decomposition (SVD)," in *Proceeding of International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*, 2005: IEEE, pp. 37–42.

[105] W. Xue, Y. Luo, Y. Yang, and Y. Huang, "Noise suppression for GPR data based on SVD of window-length-optimized hankel matrix," *Sensors*, vol. 19, no. 17, p.3807, 2019.

[106] B. Karlsen, J. Larsen, H. B. D. Sorensen, and K. B. Jakobsen, "Comparison of PCA and ICA based clutter reduction in GPR systems foranti-personal landmine detection," in *Proceeding of IEEE Signal Processing Workshop Statistical Signal Processing*, 2001: IEEE, pp. 146–149.

[107] G. Chen, L. Fu, K. Chen, C. D. Boateng, and S. Ge, "Adaptive ground clutter reduction in ground-penetrating radar data based on principal component analysis," *Transactions on Geoscience and Remote Sensing*, vol. 57, no. 6, pp. 3271–3282, 2019.

[108] D. Kumlu and I. Erer, "Clutter removal in GPR images using nonnegative matrix factorization," *Journal of Electromagnetic Waves and Applications*, vol. 32, no. 16, pp. 2055–2066, 2018.

[109] D. Kumlu and I. Erer, "Improved clutter removal in GPR by robust nonnegative matrix factorization," *IEEE Geoscience and Remote Sensing Letter*, vol. 17, no. 6, pp. 958–962, 2020.

[110] W. Lei, F. Hou, J. Xi, Q. Tan, M. Xu, X. Jiang, G. Liu, and Q. Gu, "Automatic hyperbola detection and fitting in GPR B-scan image," *Automation in Construction*, 106, p.102839, 2019.

[111] H. Liu, C. Lin, J. Cui, L. Fan, X. Xie, and B. F. Spencer, "Detection and localization of rebar in concrete by deep learning using ground penetrating radar," *Automation in Construction*, vol. 118, p.103279, 2020.

[112] H. H. Sun, Y. H. Lee, Q. Dai, C. Li, G. Ow, M. L. M. Yusof, and A. C. Yucel, "Estimating parameters of the tree root in heterogeneous doil environments via mask-guided multi-polarimetric integration neural network," *IEEE Transaction on Geoscience and Remote Sensing*, vol. 60, pp.1-16, 2021.

[113] Q. Dai, Y. H. Lee, H. H. Sun, G. Ow, M. L. M. Yusof, and A. C. Yucel, "DMRF-UNet: A two-stage deep learning scheme for GPR data inversion under heterogeneous soil conditions," *IEEE Transaction on Antennas and Propagation*, vol. 70, no. 8, pp. 6313-6328, 2022.

[114] E. Temlioglu and I. Erer, "A novel convolutional autoencoder-based clutter removal method for buried threat detection in ground-penetrating radar," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp.1-13, 2021.

[115] H. H. Sun, W. Cheng, and Z. Fan, "Learning to remove clutter in real-world GPR images using hybrid data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-14, 2022.

[116] Y. Wang, H. Qin, Y. Tang, D. Zhang, D. Yang, C. Qu, and T. Geng, "RCE-GAN: A rebar clutter elimination network to improve tunnel lining void detection from GPR images," *Remote Sensing*, vol. 14, no. 2, p. 251, 2022.

[117] Z. K. Ni, C. Shi, J. Pan, Z. Zheng, S. Ye, and G. Fang, "Declutter-GAN: GPR B-scan data clutter removal using conditional generative adversarial nets," IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 1-5, 2022.

[118] S. Anwar and N. Barnes, "Real image denoising with feature attention," in *Proceedings of IEEE/CVF International Conference on Computer Vision*, 2019: IEEE, pp. 3155–3164.

[119] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016: IEEE, pp. 770-778.

[120] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018: IEEE, pp. 7132-7141.

[121] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning dense volumetric segmentation from sparse annotation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2016: Springer, pp. 424-432.

[122] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Procedings of the International Conference on Machine Learning*, 2015: PMLR, pp. 448-456.

[123] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, N. Gimelshein, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of Advances in Neural Information Processing Systems*, vol. 32, pp. 8026-8037, 2019.

[124] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38-48, 2016.

[125] I. Giannakis, A. Giannopoulos, and C. Warren, "Realistic FDTD GPR antenna models optimized using a novel linear/nonlinear full-waveform inversion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 3, pp. 1768-1778, 2018.

[126] X. Li, Y. Grandvalet, F. Davoine, "Explicit inductive bias for transfer learning with convolutional networks," in *Proceedings of International Conference on Machine Learning*, 2018, pp. 2825-2834.

[127] B. Liu, Y. Cai, Y. Guo, X. Chen, "TransTailor: Pruning the pre-trained model for improved transfer learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, vol. 35, no. 10, pp. 8627-8634.

[128] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, R. Feris, "SpotTune: Transfer learning through adaptive fine-tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4805-4814.

[129] Y. Guo, Y. Li, L. Wang, T. Rosing, "AdaFilter: Adaptive filter fine-tuning for deep transfer learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, vol. 34, no. 4, pp. 4060-4066.

[130] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of Advances in Neural Information Processing Systems*, 2017, pp. 5998-6008.

[131] R. Tao, Z. Pan, R. K. Das, X. Qian, M. Z. Shou, and H. Li, "Is someone speaking? Exploring long-term temporal features for audio-visual active speaker detection," in *Proceedings of the ACM International Conference on Multimedia*, 2021, pp. 3927-3935.

[132] Y. Cheng, R. Wang, Z. Pan, and R. Feng, and Y. Zhang, "Look, listen, and attend: Co-attention network for self-supervised audio-visual representation learning," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 3884-3892.

[133] P. Zhang, Y. Hu, Y. Jin, S. Deng, X. Wu, and J. Chen, "A Maxwell's equations based deep learning method for time domain electromagnetic simulations," *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, vol. 6, pp. 35-40, 2021.

[134] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. *MIT Press*, 2016.

[135] H.-H. Sun, Y. H. Lee, Q. Dai, C. Li, G. Ow, M. L. M. Yusof, and A. C. Yucel, "Estimating parameters of the tree root in heterogeneous soil environments via mask-guided multi-polarimetric integration neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-16, 2022.

Appendix

A. Implementation codes for 2D GPR Data Inversion

1) MATLAB codes for 2D dataset generation under heterogeneous soil conditions.

```
%%%% Input file generation

N = 8000;
N_obj = 1;
N_type = 4;
N_soil = 10;

items = zeros(N, N_obj);
interval_permi = [2 32];

str_func = ["[x_pos_cir, y_pos_cir, rad_cir, str_cir] = circle();", ...
    "[x_pos_tri, y_pos_tri, rad_tri, ang0_tri, str_tri] = triangle();", ...
    "[x_pos_sc, y_pos_sc, rad_sc, ang0_sc, str_sc] = semi_circle();", ...
    "[x_pos_rec, y_pos_rec, l_rec, w_rec, ang_rec, str_rec] = box();"];

str_comm = ["fprintf(fid,'%1s \n', str_cir);", ...
    "fprintf(fid,'%1s \n', str_tri);", ...
    "fprintf(fid,'%1s \n', str_sc);", ...
    "fprintf(fid,'%1s \n', str_rec);"];

for i = 1:N

    item = randperm(N_type,N_obj);
    items(i,:) = item;
    seed_soil = randperm(N_soil, 1);
    permi_cir = interval_permi(1)+(rand(1,1)*(interval_permi(2)-interval_permi(1)));
    permi_tri = interval_permi(1)+(rand(1,1)*(interval_permi(2)-interval_permi(1)));
    permi_sc = interval_permi(1)+(rand(1,1)*(interval_permi(2)-interval_permi(1)));
    permi_rec = interval_permi(1)+(rand(1,1)*(interval_permi(2)-interval_permi(1)));

    [x_pos_cir, y_pos_cir, rad_cir] = deal(0);
    [x_pos_tri, y_pos_tri, rad_tri, ang0_tri] = deal(0);
    [x_pos_sc, y_pos_sc, rad_sc, ang0_sc] = deal(0);
    [x_pos_rec, y_pos_rec, l_rec, w_rec, ang_rec] = deal(0);


    for j=1:N_obj
        eval(str_func(item(j)));
    end

    fid = fopen(['input',num2str(N_obj),'/',num2str(i),'input.in'], 'w');

    fprintf(fid,'%1s \n','#domain: 1.5 0.65 0.0025');
    fprintf(fid,'%1s \n','#dx_dy_dz: 0.0025 0.0025 0.0025');
```

```
45.     fprintf(fid,'%1s \n','#time_window: 20e-9');
46.     fprintf(fid,'%1s \n',' ');
47.     fprintf(fid,'%1s \n','#waveform: gaussian 1 1e9 my_gaussian');

48.     fprintf(fid,'%1s \n','#hertzian_dipole: z 0.05 0.6 0 my_gaussia
    n');
49.     fprintf(fid,'%1s \n','#rx: 0.25 0.6 0');
50.     fprintf(fid,'%1s \n','#src_steps: 0.025 0 0');
51.     fprintf(fid,'%1s \n','#rx_steps: 0.025 0 0');
52.     fprintf(fid,'%1s \n',' ');
53.     fprintf(fid,'%1s \n','#soil_peplinski: 0.5 0.5 2.0 2.66 0.001 0
    .2 my_soil');
54.     fprintf(fid,'%1s%1s \n','#fractal_box: 0 0 0 1.5 0.5 0.0025 1.5
     1 1 1 20 my_soil my_soil_box ',num2str(seed_soil));
55.     fprintf(fid,'%1s \n',' ');
56.     fprintf(fid,'%1s%1s%1s \n','#material: ',num2str(permi_cir),' 0
     1 0 my_root_cir');
57.     fprintf(fid,'%1s%1s%1s \n','#material: ',num2str(permi_tri),' 0
     1 0 my_root_tri');
58.     fprintf(fid,'%1s%1s%1s \n','#material: ',num2str(permi_sc),' 0
    1 0 my_root_sc');
59.     fprintf(fid,'%1s%1s%1s \n','#material: ',num2str(permi_rec),' 0
     1 0 my_root_rec');
60.
61.     for j=1:N_obj
62.         eval(str_comm(item(j)));
63.     end
64.
65.     fclose(fid);
66.
67. end
68.
69. #### Circle object
70.
71. function [x_pos_cir, y_pos_cir, rad_cir, str_cir] = circle()
72.
73. interval_rad_cir = [0.05 0.08];
74. interval_x_pos_cir = [0.25 1.25];
75. interval_y_pos_cir = [0.25 0.4];
76.
77. x_pos_cir = interval_x_pos_cir(1)+(rand(1,1)*(interval_x_pos_cir(2)
    -interval_x_pos_cir(1)));
78. y_pos_cir = interval_y_pos_cir(1)+(rand(1,1)*(interval_y_pos_cir(2)
    -interval_y_pos_cir(1)));
79. rad_cir = interval_rad_cir(1)+(rand(1,1)*(interval_rad_cir(2)-
    interval_rad_cir(1)));
80.
81. str_cir = ['#cylinder: ',num2str(x_pos_cir),' ',num2str(y_pos_cir),
    ' 0 ',num2str(x_pos_cir),' ',num2str(y_pos_cir),' 0.0025 ',num2str(
    rad_cir),' my_root_cir'];
82.
83. end
84.
85. #### Box object
86. function [x_pos_rec, y_pos_rec, l_rec, w_rec, ang_rec, str_rec] = b
    ox()
87.
88. interval_x_pos_rec = [0.5 1];
89. interval_y_pos_rec = [0.25 0.3];
90. interval_l_rec = [0.12 0.16];
91. interval_w_rec = [0.04 0.06];
92. interval_ang_rec = [0 pi];
93.
94. x_pos_rec = interval_x_pos_rec(1)+(rand(1,1)*(interval_x_pos_rec(2)
    -interval_x_pos_rec(1)));
95. y_pos_rec = interval_y_pos_rec(1)+(rand(1,1)*(interval_y_pos_rec(2)
    -interval_y_pos_rec(1)));
```

```
96. l_rec = interval_l_rec(1)+(rand(1,1)*(interval_l_rec(2)-
    interval_l_rec(1)));
97. w_rec = interval_w_rec(1)+(rand(1,1)*(interval_w_rec(2)-
    interval_w_rec(1)));
98. ang_rec = interval_ang_rec(1)+(rand(1,1)*(interval_ang_rec(2)-
    interval_ang_rec(1)));
99.
100.str_rec = ['#cylinder: ',num2str(x_pos_rec),' ',num2str(y_pos_rec),
    ' 0.0025 ', ...
101.    num2str(x_pos_rec+l_rec*cos(ang_rec)),' ', num2str(y_pos_rec+l_
    rec*sin(ang_rec)),' 0.0025 ',num2str(w_rec),' my_root_rec'];
102.
103.end
104.
105.#### Semi-circle object
106.function [x_pos_sc, y_pos_sc, rad_sc, ang0_sc, str_sc] = semi_circl
    e()
107.
108.interval_rad_sc = [0.05 0.08];
109.interval_x_pos_sc = [0.25 1.25];
110.interval_y_pos_sc = [0.25 0.4];
111.interval_ang0 = [0 360];
112.
113.
114.x_pos_sc = interval_x_pos_sc(1)+(rand(1,1)*(interval_x_pos_sc(2)-
    interval_x_pos_sc(1)));
115.y_pos_sc = interval_y_pos_sc(1)+(rand(1,1)*(interval_y_pos_sc(2)-
    interval_y_pos_sc(1)));
116.rad_sc = interval_rad_sc(1)+(rand(1,1)*(interval_rad_sc(2)-
    interval_rad_sc(1)));
117.ang0_sc = interval_ang0(1)+(rand(1,1)*(interval_ang0(2)-
    interval_ang0(1)));
118.
119.str_sc = ['#cylindrical_sector: z ',num2str(x_pos_sc),' ',num2str(y
    _pos_sc),' 0 0.0025 ', ...
120.    num2str(rad_sc),' ',num2str(ang0_sc),' 180 my_root_sc'];
121.
122.end
123.
124.#### Triangle object
125.function [x_pos_tri, y_pos_tri, rad_tri, ang0_tri, str_tri] = trian
    gle()
126.
127.interval_x_pos_rec = [0.25 1.25];
128.interval_y_pos_rec = [0.25 0.4];
129.interval_rad_tri = [0.05 0.08];
130.interval_ang0_tri = [0 2*pi];
131.
132.x_pos_tri = interval_x_pos_rec(1)+(rand(1,1)*(interval_x_pos_rec(2)
    -interval_x_pos_rec(1)));
133.y_pos_tri = interval_y_pos_rec(1)+(rand(1,1)*(interval_y_pos_rec(2)
    -interval_y_pos_rec(1)));
134.rad_tri = interval_rad_tri(1)+(rand(1,1)*(interval_rad_tri(2)-
    interval_rad_tri(1)));
135.ang0_tri = interval_ang0_tri(1)+(rand(1,1)*(interval_ang0_tri(2)-
    interval_ang0_tri(1)));
136.
137.x1 = x_pos_tri + rad_tri*cos(ang0_tri);
138.y1 = y_pos_tri + rad_tri*sin(ang0_tri);
139.
140.x2 = x_pos_tri + rad_tri*cos(ang0_tri + 2*pi/3);
141.y2 = y_pos_tri + rad_tri*sin(ang0_tri + 2*pi/3);
142.
143.x3 = x_pos_tri + rad_tri*cos(ang0_tri + 2*2*pi/3);
144.y3 = y_pos_tri + rad_tri*sin(ang0_tri + 2*2*pi/3);
145.
```

```
str_tri = ['#triangle: ',num2str(x1),' ',num2str(y1),' 0 ',num2str(x2),' ',num2str(y2),' 0 ', ...
    num2str(x3),' ',num2str(y3),' 0 0.0025 my_root_tri'];

end

#### Simulation running commands generator

fid = fopen(['commands/',num2str(N_obj),'.bat'],'w');

for i = 1:N
    fprintf(fid,'%1s%1s%1s%1s%1s \n','python -m gprMax input', num2str(N_obj),'/',num2str(i),'input.in -n 48 -gpu');
    fprintf(fid,'%1s%1s%1s%1s%1s \n','python -m tools.outputfiles_merge input',num2str(N_obj),'/',num2str(i),'input --remove-files');
    fprintf(fid,'%1s \n','  ');
end

fclose(fid);
```

2) Python codes for DMRF-UNet.

```
#### DMRF-UNet.py

import os
import sys
import random
import numpy as np
import scipy.io as scio
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from sklearn import preprocessing
from sklearn.preprocessing import scale
import tensorflow as tf
from tensorflow import keras

N_class = 2
N_train = [7200, 9000]
N_test = [800, 1000]
image_sizeX = 128
image_sizeY = 128
num_channel = 1
train_data = []
train_mask1 = []
train_mask2 = []
test_data = []
test_mask1 = []
test_mask2 = []

# Load data
for i in range(N_class):
    for j in range(N_train[i]):
        x = mpimg.imread('./dataset/data/%d/%d.png'%(i+1, j+1))
        x = x.reshape(image_sizeX,image_sizeY,1)
        train_data.append(x)
        m1 = mpimg.imread('./dataset/mask1/%d/%d.png'%(i+1, j+1))
        m1 = m1.reshape(image_sizeX,image_sizeY,1)
        train_mask1.append(m1)
        m2 = mpimg.imread('./dataset/mask2/%d/mask_%d.png'%(i+1, j+1))
        m2 = m2.reshape(image_sizeX,image_sizeY,1)
```

```
40.         train_mask2.append(m2)
41.
42.     for j in range(N_train[i], N_train[i]+N_test[i]):
43.         x = mpimg.imread('./dataset/data/%d/%d.png'%(i+1, j+1))
44.         x = x.reshape(image_sizeX,image_sizeY,1)
45.         test_data.append(x)
46.         m1 = mpimg.imread('./dataset/mask1/%d/%d.png'%(i+1, j+1))
47.         m1 = m1.reshape(image_sizeX,image_sizeY,1)
48.         test_mask1.append(m1)
49.         m2 = mpimg.imread('./dataset/mask2/%d/mask_%d.png'%(i+1, j+
    1))
50.         m2 = m2.reshape(image_sizeX,image_sizeY,1)
51.         test_mask2.append(m2)
52.
53. train_data = np.array(train_data)
54. train_mask1 = np.array(train_mask1)
55. train_mask2 = np.array(train_mask2)
56. test_data = np.array(test_data)
57. test_mask1 = np.array(test_mask1)
58. test_mask2 = np.array(test_mask2)
59.
60. # Build model
61. def down_block1(x, filters, kernel_size=(3, 3), padding="same", str
    ides=1):
62.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides, activation="relu")(x)
63.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides, activation="relu")(c)
64.     p = keras.layers.MaxPool2D((2, 2), (2, 2))(c)
65.     return c, p
66.
67. def up_block1(x, skip, filters, kernel_size=(3, 3), padding="same",
     strides=1):
68.     us = keras.layers.UpSampling2D((2, 2))(x)
69.     us = keras.layers.Conv2D(filters, (2, 2), padding=padding, stri
    des=strides, activation="relu")(us)
70.     concat = keras.layers.Concatenate()([us, skip])
71.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides, activation="relu")(concat)
72.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides, activation="relu")(c)
73.     return c
74.
75. def bottleneck1(x, filters, kernel_size=(3, 3), padding="same", str
    ides=1):
76.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides, activation="relu")(x)
77.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides, activation="relu")(c)
78.     return c
79.
80. def multi_scale(x, filters, padding="same", strides=1):
81.     c1 = keras.layers.Conv2D(filters, (1,1), padding=padding, strid
    es=strides, activation="relu")(x)
82.     c3 = keras.layers.Conv2D(filters, (3,3), padding=padding, strid
    es=strides, activation="relu")(x)
83.     c5 = keras.layers.Conv2D(filters, (3,3), padding=padding, strid
    es=strides, activation="relu")(x)
84.     c5 = keras.layers.Conv2D(filters, (3,3), padding=padding, strid
    es=strides, activation="relu")(c5)
85.     c7 = keras.layers.Conv2D(filters, (3,3), padding=padding, strid
    es=strides, activation="relu")(x)
86.     c7 = keras.layers.Conv2D(filters, (3,3), padding=padding, strid
    es=strides, activation="relu")(c7)
87.     c7 = keras.layers.Conv2D(filters, (3,3), padding=padding, strid
    es=strides, activation="relu")(c7)
88.     c = keras.layers.Concatenate()([c1, c3, c5, c7])
```

```
    return c

def down_block2(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    f = int(filters/4)
    c = multi_scale(x, f)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    c = multi_scale(c, f)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    p = keras.layers.MaxPool2D((2, 2), (2, 2))(c)
    return c, p

def up_block2(x, skip, filters, kernel_size=(3, 3), padding="same", strides=1):
    f = int(filters/4)
    us = keras.layers.UpSampling2D((2, 2))(x)
    us = keras.layers.Conv2D(filters, (2, 2), padding='same', strides=1, activation="relu")(us)
    c = keras.layers.Concatenate()([us, skip])
    c = multi_scale(c, f)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    c = multi_scale(c, f)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    return c

def bottleneck2(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    f = int(filters/4)
    c = multi_scale(x, f)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    c = multi_scale(c, f)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    return c

def DMRF_UNet():

    f0 = 64
    f = [f0, f0*2, f0*4, f0*8, f0*16]
    inputs = keras.layers.Input((image_sizeX, image_sizeY, 1))

    # model_1
    p0 = inputs
    c1, p1 = down_block2(p0, f[0])
    c2, p2 = down_block2(p1, f[1])
    c3, p3 = down_block2(p2, f[2])
    c4, p4 = down_block2(p3, f[3])

    bn = bottleneck2(p4, f[4])

    u1 = up_block2(bn, c4, f[3])
    u2 = up_block2(u1, c3, f[2])
    u3 = up_block2(u2, c2, f[1])
    u4 = up_block2(u3, c1, f[0])

    output_1 = keras.layers.Conv2D(1, (1, 1), padding="same", activation="relu")(u4)
    model_1 = tf.keras.Model(inputs, output_1)

    # model_2
    pp0 = keras.layers.Concatenate()([output_1, inputs])
    cc1, pp1 = down_block2(pp0, f[0])
```

```
    cc2, pp2 = down_block2(pp1, f[1])
    cc3, pp3 = down_block2(pp2, f[2])
    cc4, pp4 = down_block2(pp3, f[3])

    bnn = bottleneck2(pp4, f[4])

    uu1 = up_block2(bnn, cc4, f[3])
    uu2 = up_block2(uu1, cc3, f[2])
    uu3 = up_block2(uu2, cc2, f[1])
    uu4 = up_block2(uu3, cc1, f[0])

    output_2 = keras.layers.Conv2D(1, (1, 1), padding="same", activ
ation="elu")(uu4)
    model_2 = tf.keras.Model(inputs, [output_1, output_2])

    return model_2

# Training
def get_lr_metric(optimizer):
    def lr(y_true, y_pred):
        return optimizer.lr
    return lr

model = DMRF_UNet()
model.summary()

total_epoch = 100
batch_size = 100
alpha = 10
beta = 1
path = '/'
model_path = path + 'exp/model.h5'
Adam = keras.optimizers.Adam(lr=1e-4)
lr_metric = get_lr_metric(Adam)
model.compile(optimizer=Adam, loss=['mse', 'mse'], loss_weights=[al
pha, beta], metrics=['mae'])
model_checkpoint = keras.callbacks.ModelCheckpoint(model_path, moni
tor='val_loss', verbose=1, save_best_only=True)
lr_checkpoint = keras.callbacks.ReduceLROnPlateau(monitor='val_loss
', factor=0.95, patience=2, min_lr=0)
history = model.fit(x=train_data, y=[train_mask1, train_mask2], bat
ch_size=batch_size, epochs=total_epoch, verbose=2, \
    validation_data=(test_data, [test_mask1, test_mask2]), callback
s=[model_checkpoint])

# Testing
model.load_weights(model_path)
model.evaluate(x=test_data, y=[test_mask1, test_mask2], batch_size=
batch_size)
[test_pred1, test_pred2] = model.predict(test_data)
for i in range(len(test_data)):
    pred1 = test_pred1[i].reshape(image_sizeX, image_sizeY) * 255
    pred1 = Image.fromarray(pred1)
    pred1.convert('L').save(path + 'exp/visual/1pred_%d.png'%(i+1))

    pred2 = test_pred2[i].reshape(image_sizeX, image_sizeY) * 255
    pred2 = Image.fromarray(pred2)
    pred2.convert('L').save(path + 'exp/visual/2pred_%d.png'%(i+1))
```

B. Implementation codes for 3D GPR Data Inversion

1) MATLAB codes for 3D dataset generation under homogeneous soil conditions.

```
#### Input file generation

N1 = 2000;
N2 = 12;
N_type = 3;
N_obj = 1;

str_func = ["[material, object, pt] = cylinder(k);", "[material, object, pt] = sphere(k);", "[material, object, pt] = box(k);"];
items = zeros(N1, N_obj);
pts = zeros(N1, N_obj);
materials = strings(N1, N_obj);
objects = strings(N1, N_obj);

for i = 1:N1

    for k = 1:N_obj
        item = randperm(N_type, 1);
        disp(item);
        items(i,k) = item;
        eval(str_func(item));
        pts(i,k) = pt;
        materials(i,k) = material;
        objects(i,k) = object;
    end

    for j = 1:N2

        y_scan = 0.02 + 0.0875 * (j - 1);

        fid = fopen(['input_files/',num2str(N_obj),'/',num2str(i),'_',num2str(j),'input.in'],'w');

        fprintf(fid,'%1s \n','#domain: 1 1 0.3');
        fprintf(fid,'%1s \n','#dx_dy_dz: 0.0025 0.0025 0.0025');
        fprintf(fid,'%1s \n','#time_window: 15e-9');
        fprintf(fid,'%1s \n','#pml_cells: 4 4 4 4 4 4');
        fprintf(fid,'%1s \n','  ');
        fprintf(fid,'%1s \n','#waveform: ricker 1 1e9 my_ricker');

        fprintf(fid,'%1s%1s%1s \n','#hertzian_dipole: y 0.02 ',num2str(y_scan),' 0.28 my_ricker');
        fprintf(fid,'%1s%1s%1s \n','#rx: 0.12 ',num2str(y_scan),' 0.28');
        fprintf(fid,'%1s \n','  ');
        fprintf(fid,'%1s \n','#src_steps: 0.0875 0 0');
        fprintf(fid,'%1s \n','#rx_steps: 0.0875 0 0');
        fprintf(fid,'%1s \n','  ');
        fprintf(fid,'%1s \n','#material: 4 0 1 0 my_soil');
        fprintf(fid,'%1s \n','#box: 0 0 0 1 1 0.26 my_soil');

        for k = 1:N_obj
            fprintf(fid,'%1s \n', materials(i,k));
            fprintf(fid,'%1s \n', objects(i,k));
        end

        fprintf(fid,'%1s \n','  ');

        if j==1
            fprintf(fid,'%1s%1s \n','#geometry_objects_write: 0 0 0 1 1 0.26 model_',num2str(i));
        end

        fclose(fid);

    end
end
```

```
62.
63. save(['params/',num2str(N_obj),'/items.mat'],'items');
64. save(['params/',num2str(N_obj),'/pts.mat'],'pts');
65.
66. #### Cylinder object
67. function [material, object, pt] = cylinder(item)
68. % Generate a cylinder
69.
70.     interval_x1_pos = [0.39 0.61];
71.     interval_y1_pos = [0.39 0.61];
72.     interval_z1_pos = [0.07 0.19];
73.     interval_x2_pos = [0.39 0.61];
74.     interval_y2_pos = [0.39 0.61];
75.     interval_z2_pos = [0.07 0.19];
76.     interval_rad = [0.02 0.05];
77.
78.     x1_pos = interval_x1_pos(1)+(rand(1)*(interval_x1_pos(2)-
    interval_x1_pos(1)));
79.     y1_pos = interval_y1_pos(1)+(rand(1)*(interval_y1_pos(2)-
    interval_y1_pos(1)));
80.     z1_pos = interval_z1_pos(1)+(rand(1)*(interval_z1_pos(2)-
    interval_z1_pos(1)));
81.     x2_pos = interval_x2_pos(1)+(rand(1)*(interval_x2_pos(2)-
    interval_x2_pos(1)));
82.     y2_pos = interval_y2_pos(1)+(rand(1)*(interval_y2_pos(2)-
    interval_y2_pos(1)));
83.     z2_pos = interval_z2_pos(1)+(rand(1)*(interval_z2_pos(2)-
    interval_z2_pos(1)));
84.     rad = interval_rad(1)+(rand(1)*(interval_rad(2)-
    interval_rad(1)));
85.     pt = randperm(19,1) + 7 + rand(1);
86.
87.     material = ['#material: ',num2str(pt),' 0 1 0 my_material_', nu
    m2str(item)];
88.     object = ['#cylinder: ',num2str(x1_pos),' ',num2str(y1_pos),' '
    ,num2str(z1_pos),' ',num2str(x2_pos),' ',num2str(y2_pos),' ',num2st
    r(z2_pos),' ',num2str(rad),' my_material_', num2str(item)];
89.
90. end
91.
92. #### Box object
93. function [material, object, pt] = box(item)
94. % Generate a cuboid/cube
95.
96.     interval_x1_pos = [0.34 0.56];
97.     interval_y1_pos = [0.34 0.56];
98.     interval_z1_pos = [0.02 0.14];
99.
100.    interval_x = [0.04 0.1];
101.    interval_y = [0.04 0.1];
102.    interval_z = [0.04 0.1];
103.
104.    x1_pos = interval_x1_pos(1)+(rand(1)*(interval_x1_pos(2)-
    interval_x1_pos(1)));
105.    y1_pos = interval_y1_pos(1)+(rand(1)*(interval_y1_pos(2)-
    interval_y1_pos(1)));
106.    z1_pos = interval_z1_pos(1)+(rand(1)*(interval_z1_pos(2)-
    interval_z1_pos(1)));
107.
108.    x = interval_x(1)+(rand(1)*(interval_x(2)-interval_x(1)));
109.    y = interval_y(1)+(rand(1)*(interval_y(2)-interval_y(1)));
110.    z = interval_z(1)+(rand(1)*(interval_z(2)-interval_z(1)));
111.
112.    pt = randperm(19,1) + 7 + rand(1);
113.
114.    material = ['#material: ',num2str(pt),' 0 1 0 my_material_', nu
    m2str(item)];
```

```
    object = ['#box: ',num2str(x1_pos),' ',num2str(y1_pos),' ',num2
    str(z1_pos),' ',num2str(x1_pos+x),' ',num2str(y1_pos+y),' ',num2str
    (z1_pos+z),' my_material_', num2str(item)];

end


#### Sphere object
function [material, object, pt] = sphere(item)
% Generate a sphere

    interval_x1_pos = [0.39 0.61];
    interval_y1_pos = [0.39 0.61];
    interval_z1_pos = [0.07 0.19];
    interval_rad = [0.02 0.05];

    x1_pos = interval_x1_pos(1)+(rand(1)*(interval_x1_pos(2)-
    interval_x1_pos(1)));
    y1_pos = interval_y1_pos(1)+(rand(1)*(interval_y1_pos(2)-
    interval_y1_pos(1)));
    z1_pos = interval_z1_pos(1)+(rand(1)*(interval_z1_pos(2)-
    interval_z1_pos(1)));
    rad = interval_rad(1)+(rand(1)*(interval_rad(2)-
    interval_rad(1)));
    pt = randperm(19,1) + 7 + rand(1);

    material = ['#material: ',num2str(pt),' 0 1 0 my_material_', nu
    m2str(item)];
    object = ['#sphere: ',num2str(x1_pos),' ',num2str(y1_pos),' ',n
    um2str(z1_pos),' ',num2str(rad),' my_material_', num2str(item)];

end

#### Simulation running commands generation
fid = fopen(['commands/',num2str(N_obj),'.bat'],'w');

for i=1:N1
    for j=1:N2
        fprintf(fid,'%1s%1s%1s%1s%1s%1s%1s \n','python -
    m gprMax input_files/',num2str(N_obj),'/',num2str(i),'_',num2str(j)
    ,'input.in -n 10 -gpu');
        fprintf(fid,'%1s%1s%1s%1s%1s%1s%1s \n','python -
    m tools.outputfiles_merge input_files/',num2str(N_obj),'/',num2str(
    i),'_',num2str(j),'input --remove-files');
        fprintf(fid,'%1s \n','  ');
    end
end

fclose(fid);
```

2) MATLAB codes for 3D dataset generation under heterogeneous soil conditions.

```
#### Input file generation.
#### Note: the object generation functions and the command generati
    on for running gprMax simulation are the same as those of homogeneo
    us soil.

N1 = 100;
N2 = 12;
N_type = 3;
N_obj = 1;

str_func = ["[material, object, pt] = cylinder(k);", "[material, ob
    ject, pt] = sphere(k);", "[material, object, pt] = box(k);"];
```

```
items = zeros(N1, N_obj);
pts = zeros(N1, N_obj);
materials = strings(N1, N_obj);
objects = strings(N1, N_obj);

for i = 1:N1

    for k = 1:N_obj
        item = randperm(N_type, 1);
        disp(item);
        items(i,k) = item;
        eval(str_func(item));
        pts(i,k) = pt;
        materials(i,k) = material;
        objects(i,k) = object;
    end

    for j = 1:N2

        y_scan = 0.02 + 0.0875 * (j - 1);

        fid = fopen(['input/',num2str(N_obj),'/',num2str(i),'_',num2str(j),'input.in'],'w');

        fprintf(fid,'%1s \n','#domain: 1 1 0.3');
        fprintf(fid,'%1s \n','#dx_dy_dz: 0.0025 0.0025 0.0025');
        fprintf(fid,'%1s \n','#time_window: 15e-9');
        fprintf(fid,'%1s \n','#pml_cells: 4 4 4 4 4 4');
        fprintf(fid,'%1s \n','  ');
        fprintf(fid,'%1s \n','#waveform: ricker 1 1e9 my_ricker');

        fprintf(fid,'%1s%1s%1s \n','#hertzian_dipole: y 0.02 ',num2str(y_scan),' 0.28 my_ricker');
        fprintf(fid,'%1s%1s%1s \n','#rx: 0.12 ',num2str(y_scan),' 0.28');
        fprintf(fid,'%1s \n','  ');
        fprintf(fid,'%1s \n','#src_steps: 0.0875 0 0');
        fprintf(fid,'%1s \n','#rx_steps: 0.0875 0 0');
        fprintf(fid,'%1s \n','  ');
        fprintf(fid,'%1s \n','#soil_peplinski: 0.5 0.5 2.0 2.66 0.001 0.15 my_soil');
        fprintf(fid,'%1s \n','#fractal_box: 0 0 0 1 1 0.26 1.5 1 1 1 50 my_soil my_fractal_box 1');

        for k = 1:N_obj
            fprintf(fid,'%1s \n', materials(i,k));
            fprintf(fid,'%1s \n', objects(i,k));
        end

        fprintf(fid,'%1s \n','  ');

        if j==1
            fprintf(fid,'%1s%1s \n','#geometry_objects_write: 0 0 0 1 1 0.26 model_',num2str(i));
        end

        fclose(fid);

    end
end
```

3) Python codes for the Denoiser of 3DInvNet.

```
#### trainDenoisingNet.py
```

```
2.  import sys, time, os, tqdm, torch, argparse, warnings, glob
3.  # from torchsummary import summary
4.  import importlib
5.  import scipy.io as sio
6.
7.  from torch.utils.data import DataLoader
8.  from data_loader import DatasetLoader
9.
10. from DenoisingNet import DenoisingNet
11.
12. def main():
13.     warnings.filterwarnings("ignore")
14.
15.     parser = argparse.ArgumentParser(description = "DenoisingNet Tr
    aining")
16.
17.     # Training related parameters setting
18.     parser.add_argument('--
    model', type=str, default="InvNetModel", help='The network structur
    e')
19.     parser.add_argument('--
    lossfc', type=str, default="MSE_loss", help='The lossfunction')
20.     parser.add_argument('--
    batch_size', type=int, default=30, help='Select {} videos as one ba
    tch')
21.     parser.add_argument('--
    lr', type=float, default=0.001, help='Learning rate')
22.     parser.add_argument('--
    lr_decay', type=float, default=0.99, help='Learning rate decay rate
    ')
23.     parser.add_argument('--
    max_epoch', type=int, default=500, help='Maximum number of epochs')

24.     parser.add_argument('--
    nDataLoaderThread', type=int, default=1, help='Number of loader thr
    eads')
25.
26.     parser.add_argument('--
    id', type=str, default="test", help='The name for saving outputs')

27.     parser.add_argument('--
    pretrain', type=str, default="test", help='The name of thr pre-
    trained model')
28.
29.     parser.add_argument('--
    train_data_path', type=str, default="dataset_real/train/data",   he
    lp='Train Data path')
30.     parser.add_argument('--
    train_mask_path', type=str, default="dataset_real/train/mask",   he
    lp='Train Mask path')
31.     parser.add_argument('--
    test_data_path', type=str, default="dataset_real/test/data",   help
    ='Test Data path')
32.     parser.add_argument('--
    test_mask_path', type=str, default="dataset_real/test/mask",   help
    ='Test Mask path')
33.
34.     # Model save path and visualization result path
35.     parser.add_argument('--
    model_path', type=str, default="workspace/exp/model/",   help='Mode
    l save path load model path')
36.     parser.add_argument('--
    visualization_path', type=str, default="workspace/exp/visual/",   h
    elp='Output Visualization Path')
37.
38.     # Mode setting
```

```
    parser.add_argument('--eval', dest='eval', action='store_true', help='Eval only')
    parser.add_argument('--load_pretrain', dest='load_pretrain', action='store_true', help='Load pretrain model or not')
    parser.add_argument('--save_model', dest='save_model', action='store_true', help='Save the model or not')
    parser.add_argument('--visual', dest='visual', action='store_true', help='Save the test result')

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Main code
    ## ===== ===== ===== ===== ===== ===== ===== =====
    args = parser.parse_args()
    args.visualization_path = args.visualization_path + args.id
    print(args.visualization_path)
    os.makedirs(args.visualization_path, exist_ok = True)

    s = DenoisingNet(**vars(args))
    print("Model para number = %.2f"%(sum(param.numel() for param in s.parameters()) / 1024 / 1024))

    train_Loader = DatasetLoader(data = args.train_data_path, mask = args.train_mask_path, **vars(args))
    train_loader = DataLoader(train_Loader, batch_size = args.batch_size, shuffle = True, num_workers = args.nDataLoaderThread, pin_memory = False)

    val_Loader = DatasetLoader(data = args.test_data_path, mask = args.test_mask_path, **vars(args))
    val_loader = DataLoader(val_Loader, batch_size = args.batch_size, shuffle = False, num_workers = args.nDataLoaderThread, pin_memory = False)


    it = 1
    decay = 0
    if args.load_pretrain == True:
        modelfiles = args.model_path + args.pretrain + "/model.model"
        print(modelfiles)
        s.loadParameters(modelfiles)
        print("Model %s loaded from previous state!"%modelfiles)

    clr = s.updateLearningRate(1)
    for ii in range(0, decay):
        clr = s.updateLearningRate(args.lr_decay)

    if args.eval == True:
        s.validate_network(it = it, loader = val_loader, **vars(args))
        quit()

    if args.visual == True:
        s.visualization(data = args.test_data_path, path = args.visualization_path, **vars(args))
        quit()

    loss_set = []
    val_loss_set = []
    while(1):
        loss = s.train_network(it = it, learningrate = max(clr), loader = train_loader, **vars(args))
        val_loss = s.validate_network(it = it, loader = val_loader, **vars(args))
```

```
87.
88.         loss_set.append(loss)
89.         val_loss_set.append(val_loss)
90.         if len(loss_set) >= 2:
91.             if loss_set[-1]> loss_set[-2]:
92.                 clr = s.updateLearningRate(args.lr_decay)
93.                 decay += 1
94.
95.         if args.save_model == True and val_loss == min(val_loss_set
    ):
96.             s.saveParameters(args.model_path + args.id + "/model.mo
    del")
97.             s.visualization(data = args.test_data_path, path = args
    .visualization_path, **vars(args))
98.
99.         if it >= args.max_epoch or max(clr) == 0:
100.             quit()
101.
102.         it += 1
103.
104.if __name__ == '__main__':
105.    main()
106.
107.
108.#### DenoisingNet.py
109.import torch
110.import torch.nn as nn
111.import torch.nn.functional as F
112.from torch.cuda.amp import autocast as autocast
113.from torch.cuda.amp import GradScaler as GradScaler
114.import importlib, sys, time, tqdm, numpy, os, glob
115.import matplotlib.image as mpimg
116.import scipy.io as sio
117.from PIL import Image
118.
119.class DenoisingNet(nn.Module):
120.    ## ===== ===== ===== ===== ===== ===== ===== =====
121.    ## Init network
122.    ## ===== ===== ===== ===== ===== ===== ===== =====
123.    def __init__(self, batch_size, lr = 0.001, model = "DenoisingNe
    tModel", lossfc = "MAE_loss", **kwargs):
124.        super(DenoisingNet, self).__init__()
125.        DenoisingNetModel = importlib.import_module('model.' + mode
    l).__getattribute__(model)
126.        LossFc = importlib.import_module('lossfc.' + lossfc).__geta
    ttribute__(lossfc)
127.
128.        self.__S__ = DenoisingNetModel().cuda()
129.        self.__L__ = LossFc().cuda()
130.        self.__optimizer__ = torch.optim.Adam(self.parameters(), lr
     = lr)
131.
132.    ## ===== ===== ===== ===== ===== ===== ===== =====
133.    ## Train network
134.    ## ===== ===== ===== ===== ===== ===== ===== =====
135.    def train_network(self, it, learningrate, loader, **kwargs):
136.        self.train()
137.
138.        stepsize = loader.batch_size
139.        loss = 0
140.        # scaler = GradScaler()
141.
142.        for i, (data, mask) in enumerate(loader):
143.
144.            self.zero_grad()
145.
146.            # with autocast():
```

```
147.             out = self.__S__.forward(data.cuda())
148.             nloss = self.__L__.forward(out, mask)
149.
150.             nloss.backward()
151.             # scaler.scale(nloss).backward()
152.
153.             loss += nloss.detach().cpu().numpy()
154.
155.             self.__optimizer__.step()
156.             # scaler.step(self.__optimizer__)
157.             # scaler.update()
158.
159.         sys.stderr.write(time.strftime("%Y-%m-%d %H:%M:%S") + \
160.         " Training %d it, learningrate = %f."%(it, learningrate) +
    \
161.         " Processing (%d/%d): (%.3f%%) "%(i + 1, loader.__len__(),
    100 * ((i+1) / loader.__len__())) + \
162.         " Loss %f \r"%(loss / (i + 1)))
163.         sys.stderr.flush()
164.         sys.stderr.write("\n")
165.         return loss
166.
167.     def validate_network(self, it, loader, **kwargs):
168.         self.eval()
169.
170.         stepsize = loader.batch_size
171.         loss = 0
172.
173.         for i, (data, mask) in enumerate(loader):
174.             with torch.no_grad():
175.                 out = self.__S__.forward(data.cuda())
176.                 nloss = self.__L__.forward(out, mask)
177.                 loss += nloss.detach().cpu().numpy()
178.
179.         sys.stderr.write(time.strftime("%Y-%m-%d %H:%M:%S") + \
180.         " Validation %d it"%(it) + \
181.         " Processing (%d/%d): (%.3f%%) "%(i + 1, loader.__len__(),
    100 * ((i+1) / loader.__len__())) + \
182.         " Loss %f \r"%(loss / (i+1)))
183.         sys.stderr.flush()
184.         sys.stderr.write("\n")
185.         return loss
186.
187.     def visualization(self, data, path, **kwargs):
188.         sys.stdout.write('\n' + time.strftime("%Y-%m-%d %H:%M:%S")
    + ' visualization_begin. \n')
189.         sys.stderr.flush()
190.         self.eval()
191.
192.         data_list = glob.glob(data + "/*")
193.         for data_name in data_list:
194.             image = sio.loadmat(data_name)['noisy_data']
195.             data_min = -9.0
196.             data_max = 9.0
197.             image1 = (image-data_min)/(data_max-data_min)
198.             data = torch.FloatTensor(image1).unsqueeze(0)
199.             data = torch.unsqueeze(data, 0)
200.             with torch.no_grad():
201.                 out = self.__S__.forward(data.cuda()).detach().cpu(
    ).numpy()[0][0]
202.                 sio.savemat(path + '/' + (data_name.split('.')[-
    2]).split('/')[-1]+'.mat', {'pred_data': out})
203.
204.         sys.stdout.write('\n' + time.strftime("%Y-%m-%d %H:%M:%S")
    + ' visualization_end. \n')
205.         sys.stderr.flush()
206.
```

```
    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Update learning rate
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def updateLearningRate(self, alpha):

        learning_rate = []
        for param_group in self.__optimizer__.param_groups:
            param_group['lr'] = param_group['lr'] * alpha
            learning_rate.append(param_group['lr'])

        return learning_rate

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Save the model
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def saveParameters(self, path):

        torch.save(self.state_dict(), path, _use_new_zipfile_serialization=False)

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Load the model's paramters
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def loadParameters(self, path):

        self.load_state_dict(torch.load(path))
        # self_state = self.state_dict()
        # loaded_state = torch.load(path)
        # for name, param in loaded_state.items():
        #   origname = name;
        #   if name not in self_state:
        #       name = name.replace("module.", "")

        #       if name not in self_state:
        #           # print("%s is not in the model."%origname)
        #           continue

        #   if self_state[name].size() != loaded_state[origname].size():
        #       sys.stderr.write("Wrong parameter length: %s, model: %s, loaded: %s"%(origname, self_state[name].size(), loaded_state[origname].size()))
        #       continue

        #   self_state[name].copy_(param)


#### data_loader.py
import os, torch, numpy, glob
import matplotlib.image as mpimg
import scipy.io as sio

## ===== ===== ===== ===== ===== ===== ===== =====
## Load the data
## ===== ===== ===== ===== ===== ===== ===== =====
def load_data(data_name):
    image = sio.loadmat(data_name)['noisy_data']
    data_min = -9.0
    data_max = 9.0
    image1 = (image-data_min)/(data_max-data_min)
    data = torch.FloatTensor(image1)
    data1 = torch.unsqueeze(data, 0)
    return data1

def load_mask(mask_name):
    image = sio.loadmat(mask_name)['clean_data']
    mask_min = -9.0
```

```
    mask_max = 9.0
    image1 = (image-mask_min)/(mask_max-mask_min)
    mask = torch.FloatTensor(image1)
    mask1 = torch.unsqueeze(mask, 0)
    return mask1

class DatasetLoader(object):
    def __init__(self, data, mask, **kwargs):
        self.data_path = data
        self.mask_path = mask
        self.data = []
        self.mask = []
        data_list = glob.glob(self.data_path + "/*")
        mask_list = glob.glob(self.mask_path + "/*")
        for data_name in data_list:
            index = (data_name.split('.')[-2]).split('/')[-1]
            data_name = self.data_path + "/%s.mat"%(index)
            mask_name = self.mask_path + "/%s.mat"%(index)
            self.data.append(data_name)
            self.mask.append(mask_name)

    def __getitem__(self, index):
        data = load_data(self.data[index])
        mask = load_mask(self.mask[index])
        return data, mask

    def __len__(self):
        return len(self.data)


#### DenoisingNetModel.py
import torch
import torch.nn as nn
import torch.nn.functional as F

class conv_block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(conv_block, self).__init__()
        layers = [
                    nn.Conv3d(in_channels, out_channels, 3, stride=
    1, padding=1),
                    nn.ReLU(True),
                ]
        self.conv_layer = nn.Sequential(*layers)

    def forward(self, x):
        return self.conv_layer(x)

class res_conv_block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(res_conv_block, self).__init__()
        layers = [
                    nn.Conv3d(in_channels, out_channels, 3, stride=
    1, padding=1),
                    nn.ReLU(True),
                    nn.Conv3d(out_channels, out_channels, 3, stride
    =1, padding=1),
                ]
        self.body = nn.Sequential(*layers)

    def forward(self, x):
        out = self.body(x)
        return F.relu(out+x)

class attention_block(nn.Module):
    def __init__(self, channels, reduction):
        super(attention_block, self).__init__()
```

```
334.         self.ave_pool = nn.AdaptiveAvgPool3d(1)
335.         self.fc = nn.Sequential(
336.             nn.Linear(channels, channels // reduction, bias = False
    ),
337.             nn.ReLU(True),
338.             nn.Linear(channels // reduction, channels, bias = False
    ),
339.             nn.Sigmoid()
340.         )
341.
342.     def forward(self, x):
343.         b, c, _, _, _ = x.size()
344.         y = self.ave_pool(x).view(b, c)
345.         y = self.fc(y).view(b, c, 1, 1, 1)
346.         return x * y.expand_as(x)
347.
348. class DenoisingNetModel(nn.Module):
349.     def __init__(self, in_channels=1, out_channels=1, f_num=8, redu
    ction=2):
350.         super(DenoisingNetModel, self).__init__()
351.         self.feature_extraction = conv_block(in_channels, f_num)
352.         self.res_net1 = res_conv_block(f_num, f_num)
353.         self.res_net2 = res_conv_block(f_num, f_num)
354.         self.feature_attention1 = attention_block(f_num, reduction)

355.         self.res_net3 = res_conv_block(f_num, f_num)
356.         self.res_net4 = res_conv_block(f_num, f_num)
357.         self.feature_attention2 = attention_block(f_num, reduction)

358.         self.reconstruction = nn.Conv3d(f_num, out_channels, 3, 1,
    1)
359.         self.act_layer = nn.ReLU(True)
360.
361.     def forward(self, x):
362.         fe = self.feature_extraction(x)
363.
364.         r1 = self.res_net1(fe)
365.         r2 = self.res_net2(r1)
366.         a1 = self.feature_attention1(r2)
367.
368.         r3 = self.res_net3(a1)
369.         r4 = self.res_net4(r3)
370.         a2 = self.feature_attention2(r4)
371.         fr = self.reconstruction(fe + a2)
372.         return self.act_layer(x + fr)
373.
374. # device = torch.device('cuda' if torch.cuda.is_available() else 'c
    pu')
375. # t = DenoisingNetModel().to(device)
376. # print(t)
377.
378.
379. #### MAE_loss.py
380. import torch
381. import torch.nn as nn
382. import numpy
383.
384. class MAE_loss(nn.Module):
385.     def __init__(self):
386.         super().__init__()
387.
388.     def forward(self, out, mask):
389.         return torch.mean(torch.abs(out - mask.cuda()))
390.
391. #### MSE_loss.py
392. import torch
393. import torch.nn as nn
```

```
394.import numpy
395.
396.class MSE_loss(nn.Module):
397.    def __init__(self):
398.        super().__init__()
399.
400.    def forward(self, out, mask):
401.        return torch.mean(torch.pow((out - mask.cuda()), 2))
```

4) Python codes for the Inverter of 3DInvNet.

```
1.  #### trainInvNet.py
2.  import sys, time, os, tqdm, torch, argparse, warnings, glob
3.  # from torchsummary import summary
4.  import importlib
5.  import scipy.io as sio
6.
7.  from torch.utils.data import DataLoader
8.  from data_loader import DatasetLoader
9.
10. from InvNet import InvNet
11.
12. def main():
13.     warnings.filterwarnings("ignore")
14.
15.     parser = argparse.ArgumentParser(description = "InvNet Training
    ")
16.
17.     # Training related parameters setting
18.     parser.add_argument('--
    model', type=str, default="InvNetModel", help='The network structur
    e')
19.     parser.add_argument('--
    lossfc', type=str, default="MAE_loss", help='The lossfunction')
20.     parser.add_argument('--
    batch_size', type=int, default=30, help='Select {} videos as one ba
    tch')
21.     parser.add_argument('--
    lr', type=float, default=0.001, help='Learning rate')
22.     parser.add_argument('--
    lr_decay', type=float, default=0.99, help='Learning rate decay rate
    ')
23.     parser.add_argument('--
    max_epoch', type=int, default=500, help='Maximum number of epochs')

24.     parser.add_argument('--
    nDataLoaderThread', type=int, default=1, help='Number of loader thr
    eads')
25.
26.     parser.add_argument('--
    id', type=str, default="test", help='The name for saving outputs')

27.     parser.add_argument('--
    pretrain', type=str, default="test", help='The name of thr pre-
    trained model')
28.
29.     parser.add_argument('--
    train_data_path', type=str, default="dataset/train/data",   help='T
    rain Data path')
30.     parser.add_argument('--
    train_mask_path', type=str, default="dataset/train/mask",   help='T
    rain Mask path')
```

```
    parser.add_argument('--test_data_path', type=str, default="dataset/test/data",   help='Test Data path')
    parser.add_argument('--test_mask_path', type=str, default="dataset/test/mask",   help='Test Mask path')

    # Model save path and visualization result path
    parser.add_argument('--model_path', type=str, default="workspace/exp/model/",   help='Model save path load model path')
    parser.add_argument('--visualization_path', type=str, default="workspace/exp/visual/",   help='Output Visualization Path')

    # Mode setting
    parser.add_argument('--eval', dest='eval', action='store_true', help='Eval only')
    parser.add_argument('--load_pretrain', dest='load_pretrain', action='store_true', help='Load pretrain model or not')
    parser.add_argument('--save_model', dest='save_model', action='store_true', help='Save the model or not')
    parser.add_argument('--visual', dest='visual', action='store_true', help='Save the test result')

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Main code
    ## ===== ===== ===== ===== ===== ===== ===== =====
    args = parser.parse_args()
    args.visualization_path = args.visualization_path + args.id
    print(args.visualization_path)
    os.makedirs(args.visualization_path, exist_ok = True)

    s = InvNet(**vars(args))
    print("Model para number = %.2f"%(sum(param.numel() for param in s.parameters()) / 1024 / 1024))

    train_Loader = DatasetLoader(data = args.train_data_path, mask = args.train_mask_path, **vars(args))
    train_loader = DataLoader(train_Loader, batch_size = args.batch_size, shuffle = True, num_workers = args.nDataLoaderThread, pin_memory = False)

    val_Loader = DatasetLoader(data = args.test_data_path, mask = args.test_mask_path, **vars(args))
    val_loader = DataLoader(val_Loader, batch_size = args.batch_size, shuffle = False, num_workers = args.nDataLoaderThread, pin_memory = False)


    it = 1
    decay = 0
    if args.load_pretrain == True:
        modelfiles = args.model_path + args.pretrain + "/model.model"
        print(modelfiles)
        s.loadParameters(modelfiles)
        print("Model %s loaded from previous state!"%modelfiles)

    clr = s.updateLearningRate(1)
    for ii in range(0, decay):
        clr = s.updateLearningRate(args.lr_decay)

    if args.eval == True:
```

```
        s.validate_network(it = it, loader = val_loader, **vars(args))
        quit()

    if args.visual == True:
        s.visualization(data = args.test_data_path, path = args.visualization_path, **vars(args))
        quit()

    loss_set = []
    val_loss_set = []
    while(1):
        loss = s.train_network(it = it, learningrate = max(clr), loader = train_loader, **vars(args))
        val_loss = s.validate_network(it = it, loader = val_loader, **vars(args))

        loss_set.append(loss)
        val_loss_set.append(val_loss)
        if len(loss_set) >= 2:
            if loss_set[-1]> loss_set[-2]:
                clr = s.updateLearningRate(args.lr_decay)
                decay += 1

        if args.save_model == True and val_loss == min(val_loss_set):
            s.saveParameters(args.model_path + args.id + "/model.model")
            s.visualization(data = args.test_data_path, path = args.visualization_path, **vars(args))

        if it >= args.max_epoch or max(clr) == 0:
            quit()

        it += 1

if __name__ == '__main__':
    main()


#### InvNet.py
import torch
import torch.nn as nn
import torch.nn.functional as F
import importlib, sys, time, tqdm, numpy, os, glob
import matplotlib.image as mpimg
import scipy.io as sio

class InvNet(nn.Module):
    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Init network
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def __init__(self, batch_size, lr = 0.001, model = "InvNetModel", lossfc = "MSE_loss", **kwargs):
        super(InvNet, self).__init__()
        InvNetModel = importlib.import_module('model.' + model).__getattribute__(model)
        LossFc = importlib.import_module('lossfc.' + lossfc).__getattribute__(lossfc)

        self.__S__ = InvNetModel().cuda()
        self.__L__ = LossFc().cuda()
        self.__optimizer__ = torch.optim.Adam(self.parameters(), lr = lr)

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Train network
```

```
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def train_network(self, it, learningrate, loader, **kwargs):
        self.train()

        stepsize = loader.batch_size
        loss = 0

        for i, (data, mask) in enumerate(loader):

            self.zero_grad()
            out = self.__S__.forward(data.cuda())
            nloss = self.__L__.forward(out, mask)
            nloss.backward()

            loss += nloss.detach().cpu().numpy()

            self.__optimizer__.step()

        sys.stderr.write(time.strftime("%Y-%m-%d %H:%M:%S") + \
        " Training %d it, learningrate = %f."%(it, learningrate) + \
        " Processing (%d/%d): (%.3f%%) "%(i + 1, loader.__len__(), 100 * ((i+1) / loader.__len__())) + \
        " Loss %f \r"%(loss / (i + 1)))
        sys.stderr.flush()
        sys.stderr.write("\n")
        return loss

    def validate_network(self, it, loader, **kwargs):
        self.eval()

        stepsize = loader.batch_size
        loss = 0

        for i, (data, mask) in enumerate(loader):
            with torch.no_grad():
                out = self.__S__.forward(data.cuda())
                nloss = self.__L__.forward(out, mask)
                loss += nloss.detach().cpu().numpy()

        sys.stderr.write(time.strftime("%Y-%m-%d %H:%M:%S") + \
        " Validation %d it"%(it) + \
        " Processing (%d/%d): (%.3f%%) "%(i + 1, loader.__len__(), 100 * ((i+1) / loader.__len__())) + \
        " Loss %f \r"%(loss / (i+1)))
        sys.stderr.flush()
        sys.stderr.write("\n")
        return loss

    def visualization(self, data, path, **kwargs):
        sys.stdout.write('\n' + time.strftime("%Y-%m-%d %H:%M:%S") + ' visualization_begin. \n')
        sys.stderr.flush()
        self.eval()

        data_list = glob.glob(data + "/*")
        for data_name in data_list:
            image = sio.loadmat(data_name)['clean_data']
            min_d = -9.0
            max_d = 9.0
            image1 = (image-min_d)/(max_d-min_d)
            data = torch.FloatTensor(image1).unsqueeze(0)
            data = torch.unsqueeze(data, 0)
            with torch.no_grad():
                out = self.__S__.forward(data.cuda()).detach().cpu().numpy()[0][0]
```

```
                sio.savemat(path + '/' + (data_name.split('.')[-2]).split('/')[-1]+'.mat', {'pred': out})

        sys.stdout.write('\n' + time.strftime("%Y-%m-%d %H:%M:%S") + ' visualization_end. \n')
        sys.stderr.flush()

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Update learning rate
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def updateLearningRate(self, alpha):

        learning_rate = []
        for param_group in self.__optimizer__.param_groups:
            param_group['lr'] = param_group['lr'] * alpha
            learning_rate.append(param_group['lr'])

        return learning_rate

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Save the model
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def saveParameters(self, path):

        torch.save(self.state_dict(), path, _use_new_zipfile_serialization=False)

    ## ===== ===== ===== ===== ===== ===== ===== =====
    ## Load the model's paramters
    ## ===== ===== ===== ===== ===== ===== ===== =====
    def loadParameters(self, path):

        self.load_state_dict(torch.load(path))


#### data_loader.py
import os, torch, numpy, glob
import matplotlib.image as mpimg
import scipy.io as sio

## ===== ===== ===== ===== ===== ===== ===== =====
## Load the data
## ===== ===== ===== ===== ===== ===== ===== =====
def load_data(data_name):
    image = sio.loadmat(data_name)['clean_data']
    min_d = -9.0
    max_d = 9.0
    image1 = (image-min_d)/(max_d-min_d)
    data = torch.FloatTensor(image1)
    data1 = torch.unsqueeze(data, 0)
    return data1

def load_mask(mask_name):
    image = sio.loadmat(mask_name)['mask']
    mask = torch.FloatTensor(image)
    mask1 = torch.unsqueeze(mask, 0)
    return mask1

class DatasetLoader(object):
    def __init__(self, data, mask, **kwargs):
        self.data_path = data
        self.mask_path = mask
        self.data = []
        self.mask = []
        data_list = glob.glob(self.data_path + "/*")
        mask_list = glob.glob(self.mask_path + "/*")
        for data_name in data_list:
```

```
            index = (data_name.split('.')[-2]).split('/')[-1]
            data_name = self.data_path + "/%s.mat"%(index)
            mask_name = self.mask_path + "/%s.mat"%(index)
            self.data.append(data_name)
            self.mask.append(mask_name)

    def __getitem__(self, index):
        data = load_data(self.data[index])
        mask = load_mask(self.mask[index])
        return data, mask

    def __len__(self):
        return len(self.data)


#### InvNetModel_ff.py
import torch
import torch.nn as nn
import torch.nn.functional as F

class pub(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(pub, self).__init__()
        self.conv_layer1 = nn.Sequential(nn.Conv3d(in_channels, out_channels, kernel_size=3, stride=1, padding=1),
                                         nn.BatchNorm3d(out_channels),
                                         nn.ReLU(True))
        self.conv_layer2 = nn.Sequential(nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1),
                                         nn.BatchNorm3d(out_channels),
                                         nn.ReLU(True))
        self.conv_layer3 = nn.Sequential(nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1),
                                         nn.BatchNorm3d(out_channels),
                                         nn.ReLU(True))

    def forward(self, x):
        c1 = self.conv_layer1(x)
        c2 = self.conv_layer2(c1)
        c3 = self.conv_layer3(c2)
        return torch.cat((c1, c2, c3), dim=1)

class down(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(down, self).__init__()
        self.pub = pub(in_channels, out_channels)
        self.pool = nn.MaxPool3d(2, stride=2)

    def forward(self, x):
        x = self.pub(x)
        return x,self.pool(x)

class up(nn.Module):
    def __init__(self, in_channels, out_channels, sample=False):
        super(up, self).__init__()
        if sample:
            self.sample = nn.Upsample(scale_factor=2, mode='nearest')
        else:
            self.sample = nn.ConvTranspose3d(in_channels, out_channels, 2, stride=2)
        self.pub = pub(in_channels//2+out_channels, out_channels)

    def forward(self, x, x1):
        x = self.sample(x)
        x = torch.cat((x, x1), dim=1)
        x = self.pub(x)
```

```
        return x

class InvNetModel_ff(nn.Module):
    def __init__(self, in_channels=1, out_channels=1, f_num = [8, 16, 32, 64, 128], n=3, sample=False):
        super(InvNetModel_ff, self).__init__()
        self.down1 = down(in_channels, f_num[0])
        self.down2 = down(f_num[0]*n, f_num[1])
        self.down3 = down(f_num[1]*n, f_num[2])
        self.down4 = down(f_num[2]*n, f_num[3])
        self.bridge = pub(f_num[3]*n, f_num[4])
        self.up4 = up(f_num[4]*n, f_num[3], sample)
        self.up3 = up(f_num[3]*n, f_num[2], sample)
        self.up2 = up(f_num[2]*n, f_num[1], sample)
        self.up1 = up(f_num[1]*n, f_num[0], sample)
        self.con_last = nn.Conv3d(f_num[0]*3, out_channels, 1)

    def forward(self, x):
        x1,x = self.down1(x)
        x2,x = self.down2(x)
        x3,x = self.down3(x)
        x4,x = self.down4(x)
        x = self.bridge(x)
        x = self.up4(x,x4)
        x = self.up3(x,x3)
        x = self.up2(x,x2)
        x = self.up1(x,x1)
        out = self.con_last(x)
        return out


#### MAE_loss.py
import torch
import torch.nn as nn
import numpy

class MAE_loss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, out, mask):
        return torch.mean(torch.abs(out - mask.cuda()))


#### MSE_loss.py
import torch
import torch.nn as nn
import numpy

class MSE_loss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, out, mask):
        return torch.mean(torch.pow((out - mask.cuda()), 2))
```

C. Implementation codes for Fast GPR Forward Solver

1) MATLAB and Python codes for dataset generation under heterogeneous soil conditions.

```
#### triangle_generator.py
import random
import numpy as np
import cv2
from PIL import Image
import math

N1 = 0
N2 = 2750
num = 12
theta = np.linspace(0, 2 * np.pi, num)
for i in range(N1,N2):
    x = random.randint(80, 520)
    y = random.randint(50, 150)
    rad = random.randint(16, 35)
    x1 = x + rad*np.cos(theta)
    y1 = y + rad*np.sin(theta)
    p = random.sample(range(0,num-1), 3)
    p.sort()
    pts = np.array([[x1[p[0]],y1[p[0]]],[x1[p[1]],y1[p[1]]],[x1[p[2]],y1[p[2]]]], np.int32)
    # pts = pts.reshape((-1,1,2))
    img = np.zeros((200,600),np.uint8)
    cv2.fillConvexPoly(img, pts, 255)

    # cv2.imshow('image', img)
    # cv2.waitKey(10000)

    img1 = Image.fromarray(img)
    img1.convert('L').save('img/triangle/%d.png'%(i+1))


#### quadrilateral_generator.py
import random
import numpy as np
import cv2
from PIL import Image
import math

N1 = 0
N2 = 2750
num = 12
theta = np.linspace(0, 2 * np.pi, num)
for i in range(N1,N2):
    x = random.randint(80, 520)
    y = random.randint(50, 150)
    rad = random.randint(16, 35)
    x1 = x + rad*np.cos(theta)
    y1 = y + rad*np.sin(theta)
    p = random.sample(range(0,num-1), 4)
    p.sort()
    pts = np.array([[x1[p[0]],y1[p[0]]],[x1[p[1]],y1[p[1]]],[x1[p[2]],y1[p[2]]], [x1[p[3]],y1[p[3]]]], np.int32)
    img = np.zeros((200,600),np.uint8)
    cv2.fillConvexPoly(img, pts, 255)

    img1 = Image.fromarray(img)
    img1.convert('L').save('img/quadrilateral/%d.png'%(i+1))


#### pentagon_generator.py
import random
import numpy as np
import cv2
from PIL import Image
import math

```

```
N1 = 0
N2 = 2750
num = 12
theta = np.linspace(0, 2 * np.pi, num)
for i in range(N1,N2):
    x = random.randint(80, 520)
    y = random.randint(50, 150)
    rad = random.randint(16, 35)
    x1 = x + rad*np.cos(theta)
    y1 = y + rad*np.sin(theta)
    p = random.sample(range(0,num-1), 5)
    p.sort()
    pts = np.array([[x1[p[0]],y1[p[0]]],[x1[p[1]],y1[p[1]]],[x1[p[2]],y1[p[2]]], [x1[p[3]],y1[p[3]]], [x1[p[4]],y1[p[4]]]], np.int32)

    img = np.zeros((200,600),np.uint8)
    cv2.fillConvexPoly(img, pts, 255)

    img1 = Image.fromarray(img)
    img1.convert('L').save('img/pentagon/%d.png'%(i+1))


#### hexagon_generator.py
import random
import numpy as np
import cv2
from PIL import Image
import math

N1 = 0
N2 = 2750
num = 12
theta = np.linspace(0, 2 * np.pi, num)
for i in range(N1,N2):
    x = random.randint(80, 520)
    y = random.randint(50, 150)
    rad = random.randint(16, 35)
    x1 = x + rad*np.cos(theta)
    y1 = y + rad*np.sin(theta)
    p = random.sample(range(0,num-1), 6)
    p.sort()
    pts = np.array([[x1[p[0]],y1[p[0]]],[x1[p[1]],y1[p[1]]],[x1[p[2]],y1[p[2]]], [x1[p[3]],y1[p[3]]], [x1[p[4]],y1[p[4]]], [x1[p[5]],y1[p[5]]]], np.int32)
    img = np.zeros((200,600),np.uint8)
    cv2.fillConvexPoly(img, pts, 255)

    img1 = Image.fromarray(img)
    img1.convert('L').save('img/hexagon/%d.png'%(i+1))


#### circle_generator.py
import random
import numpy as np
import cv2
from PIL import Image
from matplotlib import pyplot as plt

N1 = 0
N2 = 2750
for i in range(N1,N2):
    x = random.randint(80, 520)
    y = random.randint(40, 160)
    rad = random.randint(10, 25)
    img = np.zeros((200,600),np.uint8)
    cv2.circle(img,(x,y),rad,255,-1)

```

```
    # cv2.imshow('image', img)
    # cv2.waitKey(10000)

    img1 = Image.fromarray(img)
    img1.convert('L').save('img/circle/%d.png'%(i+1))


#### ellipse_generator.py
import random
import numpy as np
import cv2
from PIL import Image
from matplotlib import pyplot as plt

N1 = 0
N2 = 2750
for i in range(N1,N2):
    x = random.randint(80, 520)
    y = random.randint(40, 160)
    rad = random.sample(range(10,25), 2)
    ang = random.randint(0, 360)
    img = np.zeros((200,600),np.uint8)
    cv2.ellipse(img,(x,y),(rad[0],rad[1]),ang,0,360,255,-1)

    # cv2.imshow('image', img)
    # cv2.waitKey(10000)

    img1 = Image.fromarray(img)
    img1.convert('L').save('img/ellipse/%d.png'%(i+1))


#### material_file_generator.m
N1 = 1;
N2 = 2750;
shape_type = ["circle","ellipse","hexagon","pentagon","quadrilatera
    l","triangle"];
shape = char(shape_type(6));
a = 1; b = 32;
c = 0; d = 0.0008;

for i = N1:N2
    pt = (b-a)*rand(1,1) + a;
    ct = (d-c)*rand(1,1) + c;
    soil_fid = fopen('soil/1soil_materials.txt','r');
    fid = fopen(['material/',shape,'/',num2str(i),'.txt'],'w');
    while feof(soil_fid)~=1
        str = fgetl(soil_fid);
        fprintf(fid,'%1s \n',str);
    end

    fprintf(fid,'%1s%1s%1s%1s%1s \n','#material: ',num2str(pt),' ',
    num2str(ct),' 1 0 my_obj');

    fclose(fid);
    fclose(soil_fid);

end


#### h5_generatpr.py
N1 = 1;
N2 = 2750;
shape_type = ["circle","ellipse","hexagon","pentagon","quadrilatera
    l","triangle"];
shape = char(shape_type(6));
x_size = 200; y_size = 600;
dx = 0.0025; dy = 0.0025; dz = 0.0025;
```

```
N_soil = 10;

for i = N1:N2
    ind = randperm(N_soil,1);
    tmp = h5read(['soil/',num2str(ind),'soil.h5'], '/data');
    soil_geo = zeros(x_size, y_size);
    soil_geo(:,:) = tmp(1,1:x_size,:);
    soil_geo = flipud(soil_geo);
    img = imread(['img/',shape,'/',num2str(i),'.png'])./255;
%     img = imresize(img, [x_size y_size], 'nearest');
    img = (1 - single(img)).*single(soil_geo);
    img(img==0) = 52;
    img = flipud(img);
    filename = ['h5/',shape,'/',num2str(i),'.h5'];
    data_size = [1 x_size y_size];
    dx_dy_dz = [dx, dy, dz];
    data = reshape(img,data_size);
    h5create(filename, '/data',data_size,'Datatype','int16');
    h5write(filename, '/data', data);
    h5writeatt(filename,'/','dx_dy_dz',dx_dy_dz);
end


#### input_file_generator
N1 = 1;
N2 = 2750;
shape_type = ["circle","ellipse","hexagon","pentagon","quadrilateral","triangle"];
shape = char(shape_type(4));

for i = N1:N2

    fid = fopen(['input/',shape,'/',num2str(i),'input.in'],'w');

    fprintf(fid,'%1s \n','#domain: 1.5 0.6 0.0025');
    fprintf(fid,'%1s \n','#dx_dy_dz: 0.0025 0.0025 0.0025');
    fprintf(fid,'%1s \n','#time_window: 20e-9');
    fprintf(fid,'%1s \n','  ');
    fprintf(fid,'%1s \n','#waveform: gaussian 1 1e9 my_pulse');
    fprintf(fid,'%1s \n','#hertzian_dipole: z 0.05 0.55 0 my_pulse');
    fprintf(fid,'%1s \n','#rx: 0.25 0.55 0');
    fprintf(fid,'%1s \n','  ');
    fprintf(fid,'%1s \n','#src_steps: 0.025 0 0');
    fprintf(fid,'%1s \n','#rx_steps: 0.025 0 0');
    fprintf(fid,'%1s \n','  ');
    fprintf(fid,'%1s%1s%1s%1s%1s%1s%1s%1s%1s \n','#geometry_objects_read: 0 0 0 h5/',shape,'/',num2str(i),'.h5 material/',shape,'/',num2str(i),'.txt');
    fclose(fid);

end


#### Command generation for running simulation with gprMax
shape_type = ["circle","ellipse","hexagon","pentagon","quadrilateral","triangle"];
shape = char(shape_type(1));

fid = fopen(['commands/commands_',shape,'.bat'],'w');
for i = N1:N2

    fprintf(fid,'%1s%1s%1s%1s%1s \n','python -m gprMax input/',shape,'/',num2str(i),'input.in -n 49 -gpu');
    fprintf(fid,'%1s%1s%1s%1s%1s \n','python -m tools.outputfiles_merge input/',shape,'/',num2str(i),'input --remove-files');
```

```
    fprintf(fid,'%1s \n','  ');
end
fclose(fid);
```

2) Python codes for the proposed network for forward solving.

```
#### ForwardSolver.py
import os
import sys
import random
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from sklearn import preprocessing
from sklearn.preprocessing import scale
import tensorflow as tf
from tensorflow import keras

N_train = [2500,2500,2500,2500,2500,2500]
N_test_start = [2500,2500,2500,2500,2500,2500]
N_test = [250,250,250,250,250,250]
N_class = len(N_train)
data_sizeX = 200
data_sizeY = 600
mask_sizeX = 256
mask_sizeY = 256
num_channel = 1
train_data1 = []
train_data2 = []
train_mask= []
test_data1 = []
test_data2 = []
test_mask = []
path = '/'
sub_path = 'exp/'

# Load data
for i in range(N_class):
    for j in range(N_train[i]):
        data1 = sio.loadmat(path+'dataset/%d/data1/%d.mat'%(i+1, j+1))['data1']
        train_data1.append(data1.reshape((data_sizeX,data_sizeY,1)))
        data2 = sio.loadmat(path+'dataset/%d/data2/%d.mat'%(i+1, j+1))['data2']
        train_data2.append(data2.reshape((data_sizeX,data_sizeY,1)))
        mask = sio.loadmat(path+'dataset/%d/mask/%d.mat'%(i+1, j+1))['mask']
        train_mask.append(mask.reshape((mask_sizeX,mask_sizeY,1)))

    for j in range(N_test_start[i], N_test_start[i]+N_test[i]):
        data1 = sio.loadmat(path+'dataset/%d/data1/%d.mat'%(i+1, j+1))['data1']
        test_data1.append(data1.reshape((data_sizeX,data_sizeY,1)))

        data2 = sio.loadmat(path+'dataset/%d/data2/%d.mat'%(i+1, j+1))['data2']
        test_data2.append(data2.reshape((data_sizeX,data_sizeY,1)))
```

```
48.         mask = sio.loadmat(path+'dataset/%d/mask/%d.mat'%(i+1, j+1)
    )['mask']
49.         test_mask.append(mask.reshape((mask_sizeX,mask_sizeY,1)))
50.
51. train_data1 = np.array(train_data1)
52. train_data2 = np.array(train_data2)
53. train_mask = np.array(train_mask)
54. test_data1 = np.array(test_data1)
55. test_data2 = np.array(test_data2)
56. test_mask = np.array(test_mask)
57.
58. # Build model
59. def cross_attention(x1, x2, filters, h=8):
60.     v1 = keras.layers.Conv2D(filters, (1,1), padding='same', stride
    s=1)(x1)
61.     q2 = keras.layers.Conv2D(filters, (1,1), padding='same', stride
    s=1)(x2)
62.     a1 = keras.layers.MultiHeadAttention(num_heads=h, key_dim=filte
    rs//h)(q2, v1)
63.     o1 = keras.layers.LayerNormalization()(a1 + v1)
64.     out1 = keras.layers.LayerNormalization()(o1 + keras.layers.Conv
    2D(filters, (1,1), padding='same', strides=1)(o1))
65.     return out1
66.
67. def down_block(x, filters, kernel_size=(3,3), padding='same', strid
    es=1):
68.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides)(x)
69.     c = keras.layers.Activation('relu')(c)
70.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides)(c)
71.     c = keras.layers.Activation('relu')(c)
72.     p = keras.layers.MaxPool2D((2, 2), (2, 2))(c)
73.     return p
74.
75. def connect_block(x, filters, kernel_size=(3,3)):
76.     c = keras.layers.Conv2D(filters, kernel_size, padding='same', s
    trides=(1,3))(x)
77.     c = keras.layers.Activation('relu')(c)
78.     c = keras.layers.Conv2DTranspose(filters, kernel_size, padding=
    'valid', strides=1)(c)
79.     c = keras.layers.Activation('relu')(c)
80.     c = keras.layers.Conv2D(filters, kernel_size, padding='same', s
    trides=1)(c)
81.     c = keras.layers.Activation('relu')(c)
82.     return c
83.
84. def up_block(x, filters, kernel_size=(3,3), padding='same', strides
    =1):
85.     us_x = keras.layers.UpSampling2D((2, 2))(x)
86.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides)(us_x)
87.     c = keras.layers.Activation('relu')(c)
88.     c = keras.layers.Conv2D(filters, kernel_size, padding=padding,
    strides=strides)(c)
89.     c = keras.layers.Activation('relu')(c)
90.     return c
91.
92. def network():
93.
94.     # Feature Map Channel
95.     f0 = 16
96.     f = [f0, f0*2, f0*4, f0*8, f0*16, f0*32]
97.     inputs1 = keras.layers.Input((data_sizeX, data_sizeY, num_chann
    el))
98.     inputs2 = keras.layers.Input((data_sizeX, data_sizeY, num_chann
    el))
```

```
99.
100.    # Encoder1
101.    p10 = inputs1
102.    p11 = down_block(p10, f[0])
103.    p12 = down_block(p11, f[1])
104.    p13 = down_block(p12, f[2])
105.    p14 = down_block(p13, f[3])
106.    e1 = down_block(p14, f[4])
107.
108.    # Encoder2
109.    p20 = inputs2
110.    p21 = down_block(p20, f[0])
111.    p22 = down_block(p21, f[1])
112.    p23 = down_block(p22, f[2])
113.    p24 = down_block(p23, f[3])
114.    e2 = down_block(p24, f[4])
115.
116.    # Fusion
117.    fu1 = cross_attention(e1, e2, f[4])
118.    fu2 = cross_attention(e2, e1, f[4])
119.    fu = connect_block(keras.layers.Concatenate()([fu1, fu2]), f[5]
    )
120.
121.    # Decoder
122.    u1 = up_block(fu, f[4])
123.    u2 = up_block(u1, f[3])
124.    u3 = up_block(u2, f[2])
125.    u4 = up_block(u3, f[1])
126.    u5 = up_block(u4, f[0])
127.
128.    outputs = keras.layers.Conv2D(1, (1, 1), padding='same')(u5)
129.    model = tf.keras.Model([inputs1, inputs2], outputs)
130.
131.    return model
132.
133.def get_lr_metric(optimizer):
134.    def learning_rate(y_true, y_pred):
135.        return optimizer.learning_rate
136.    return learning_rate
137.
138.model = network()
139.model.summary()
140.
141.# Training
142.total_epoch = 100
143.batch_size = 10
144.model_path = path+sub_path+'model.h5'
145.Adam = keras.optimizers.Adam(learning_rate=1e-4)
146.lr_metric = get_lr_metric(Adam)
147.model.compile(optimizer=Adam, loss='mse', metrics=[lr_metric])
148.model_checkpoint = keras.callbacks.ModelCheckpoint(model_path, moni
    tor='val_loss', verbose=1, save_best_only=True)
149.lr_checkpoint = keras.callbacks.ReduceLROnPlateau(monitor='val_loss
    ', factor=0.98, patience=1, min_lr=0)
150.history = model.fit(x=[train_data1,train_data2], y=train_mask, batc
    h_size=batch_size, epochs=total_epoch, verbose=2, \
151.    validation_data=([test_data1,test_data2], test_mask), callbacks
    =[model_checkpoint,lr_checkpoint])
152.
153.# Testing
154.model.load_weights(model_path)
155.model.evaluate(x=[test_data1,test_data2], y=test_mask, batch_size=b
    atch_size)
156.test_pred = model.predict([test_data1,test_data2])
157.sio.savemat(path+sub_path+'data1.mat', {'data1': test_data1})
158.sio.savemat(path+sub_path+'data2.mat', {'data2': test_data2})
159.sio.savemat(path+sub_path+'mask.mat', {'mask': test_mask})
```

```
160.sio.savemat(path+sub_path+'pred.mat', {'pred': test_pred})
```