Secure and Reliable Deep Learning in Signal Processing

Jinshan Liu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

Jung-Min (Jerry) Park, Chair
Y.Thomas Hou
Carl Dietrich
Haibo Zeng
Bert Huang

May 6, 2021
Arlington, Virginia

Keywords: Deep learning, Signal processing, Security, Reliability

Secure and Reliable Deep Learning in Signal Processing

Jinshan Liu

(ABSTRACT)

In conventional signal processing approaches, researchers need to manually extract features from raw data that can better describe the underlying problem. Such a process requires strong domain knowledge about the given problems. On the contrary, deep learning-based signal processing algorithms can discover features and patterns that would not be apparent to humans by feeding a sufficient amount of training data. In the past decade, deep learning has proved to be efficient and effective at delivering high-quality results.

Deep learning has demonstrated its great advantages in image processing and text mining. One of the most promising applications of deep learning-based signal processing techniques is autonomous driving. Today, many companies are developing and testing autonomous vehicles. High-level autonomous vehicles are expected to be commercialized in the near future. Besides, deep learning has demonstrated great potential in wireless communications applications. Researchers have addressed some of the most challenging problems such as transmitter classification and modulation recognition using deep learning.

Despite these advantages, there exist a wide range of security and reliability issues when applying deep learning models to real-world applications. First, deep learning models could not generate reliable results for testing data if the training data size is insufficient. Since generating training data is time consuming and resource intensive, it is important to understand the relationship between model reliability and the size of training data. Second, deep learning models could generate highly unreliable results if the testing data are significantly different from the training data, which we refer to as "out-of-distribution (OOD)" data.

Failing to detect OOD testing data may expose serious security risks. Third, deep learning algorithms can be easily fooled when the input data are falsified. Such vulnerabilities may cause severe risks in safety-critical applications such as autonomous driving.

In this dissertation, we focus on the security and reliability issues in deep learning models in the following three aspects. (1) We systematically study how the model performance changes as more training data are provided in wireless communications applications. (2) We discuss how OOD data can impact the performance of deep learning-based classification models in wireless communications applications. We propose FOOD (**F**eature representation for **OOD** detection), a unified model that can detect OOD testing data effectively and perform classifications for regular testing data simultaneously. (3) We focus on the security issues of applying deep learning algorithms to autonomous driving. We discuss the impact of *Perception Error Attacks (PEAs)* on LIDAR and camera and propose a countermeasure called LIFE (**L**IDAR and **I**mage data **F**usion for detecting perception **E**rrors).

Secure and Reliable Deep Learning in Signal Processing

Jinshan Liu

(GENERAL AUDIENCE ABSTRACT)

Deep learning has provided computers and mobile devices extraordinary powers to solve challenging signal processing problems. For example, current deep learning technologies are able to improve the quality of machine translation significantly, recognize speech as accurately as human beings, and even outperform human beings in face recognition.

Although deep learning has demonstrated great advantages in signal processing, it can be insecure and unreliable if the model is not trained properly or is tested under adversarial scenarios. In this dissertation, we study the following three security and reliability issues in deep learning-based signal processing methods. First, we provide insights on how the deep learning model reliability is changed as the size of training data increases. Since generating training data requires a tremendous amount of labor and financial resources, our research work could help researchers and product developers to gain insights on balancing the tradeoff between model performance and training data size. Second, we propose a novel model to detect the abnormal testing data that are significantly different from the training data. In deep learning, there is no performance guarantee when the testing data are significantly different from the training data. Failing to detect such data may cause severe security risks. Finally, we design a system to detect sensor attacks targeting autonomous vehicles. Deep learning can be easily fooled when the input sensor data are falsified. Security and safety can be enhanced significantly if the autonomous driving systems are able to figure out the falsified sensor data before making driving decisions.

Dedication

To my parents Hongwei Jin and Tongyan Liu.

Acknowledgments

First and foremost, I am extremely grateful to my supervisor Dr. Jung-Min (Jerry) Park for his invaluable advice, continuous support, and patience during my Ph.D. career. His openness provided me the opportunity to explore various research topics and get a simultaneous master's degree in mathematics. He not only instructed me with technical knowledge but also guided me on how to present and write research works attractively. Besides, he offered me the Avast data science internship opportunity to help me gain industrial experience.

Next, I would like to express my sincere gratitude to my Ph.D. committee members, Dr. Y.Thomas Hou, Dr. Carl Dietrich, Dr. Haibo Zeng, and Dr. Bert Huang, for their precious time, insightful comments and suggestions on my research works. I would also like to thank my colleges, Gaurang Naik, Taiwo Oyedare, and Hanif Rahbari, for the collaborations and deep discussions on the research works. In addition, I would like to offer my special thanks to He Li for the valuable discussions on the challenging mathematical problems when we pursue the master's degree in mathematics together.

Moreover, I would like to thank my parents for their strong support and empathy. You are always there for me. Finally, I would like to thank Xuewen Cui, Yecheng Zhao, Xiangyu Zhang, Boyu Lyu, and all the other friends for their supports and help.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

ANN Artificial Neural Network

AP Average Precision

AUPR Area Under the Precision Recall Curve

AUROC Area Under the Receiver Operating Characteristics

CNN Convolutional Neural Network

CWT Continuous Wavelet Transformation

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DNN Deep Neural Network

ELBO Evidence Lower Bound

FFT Fast Fourier Transform

FN False Negative

FOOD Feature representation for OOD detection

FP False Positive

FPR False Positive Rate

GAN Generative Adversarial Network

ID In-Distribution

IOM Intersection over Minimum

IOU Intersection over Union

IQ In-phase and Quadrature-phase

KL Kullback–Leibler

LIDAR Light Detection And RAnging

LIFE LIDAR and Image data Fusion for detecting perception Error

LOS Line-of-Sight

LSTM Long Short Term Memory

NLOS Non-Line-of-Sight

OFDM Orthogonal Frequency-Division Multiplexing

OOD Out-of-Distribution

PEA Perception Error Attack

QPSK Quadrature Phase-Shift Keying

ReLU Rectified Linear Unit

RF Radio Frequency

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

SAE Society of Automotive Engineers

SAS Spectrum Access System

SDR Software Defined Radio

SGD Stochastic Gradient Descent

SIFT Scale-Invariant Feature Transform

SNR Signal-to-Noise Ratio

SSIM Structural Similarity

TOPS Tera Operations Per Second

TP True Positive

TPR True Positive Rate

UAV Unmanned Aerial Vehicle

USRP Universal Software Radio Peripheral

V2V Vehicle to Vehicle

VAE Variational Autoencoder

VC Vapnik-Chervonenkis

YOLO You Only Look Once

Chapter 1

Introduction

In this chapter, we first give a brief introduction to the advantages of deep learning over traditional signal processing methods. Then we discuss the security and reliability issues in applying deep learning to real-world applications. Next, we introduce the target applications to conduct our experiments. Finally, we highlight our contributions to making deep machine learning-based signal processing algorithms more secure and reliable.

1.1 Deep Learning in Signal Processing

1.1.1 Advantages of Deep Learning in Signal Processing

In conventional signal processing methods, researchers need to discover and analyze features manually. This is often accomplished by building analytical models and by applying signal transformation formulas. On the contrary, deep learning can discover trends and patterns that would not be apparent to humans by reviewing large volumes of data. In recent years, deep learning techniques have enjoyed significant success in solving challenging problems in several application domains, including image classification [44], natural language translation [95], speech recognition [31], etc. More amazingly, there are several areas that deep learning outperforms human beings, such as object recognition [29] and gaming [63, 94]. Such great achievements pose a significant opportunity in signal processing automation.

1.1.2 Reliability and Security Issues in Deep Learning

Despite these great advantages in deep learning-based signal processing techniques, deep learning systems can be fragile and easily fooled. Fundamentally, deep learning is a statistical model that extracts features and patterns from training data. In real-world applications, researchers and product developers seek to build models that can generalize well to unseen testing data given a limited amount of training data. There exist security and reliability issues in both the training stage and the testing stage.

A general procedure of applying deep learning-based signal processing algorithms to real-world applications can be summarized as follows: First, define the problem and collect training data associated with the problem. The amount of training data is constrained by labor resources and financial resources. Second, build and train the deep learning models using the training data. Finally, select the most suitable model based on model performance and resource constraints (e.g., RAM, execution time, etc).

From a statistical point of view, a testing data $\mathbf{x}$ can be modeled as a data point sampled from a probability distribution $p(\mathbf{x})$. In real-world applications, $p(\mathbf{x})$ is usually very complex and cannot be expressed in an analytical form. The collection of training data is equivalent to generating samples from distribution $p(\mathbf{x})$. The objective of a deep learning model is to learn representative features that can generalize well for samples from $p(\mathbf{x})$ using a limited amount of training data. Therefore, a deep learning model works reliably on testing data if the following two conditions are satisfied: (1) the model is able to learn enough features that apply to data points sampled from $p(\mathbf{x})$ and (2) the testing data are statistically similar to the training data. Security and reliability issues may arise if either of the aforementioned conditions is not satisfied. In this dissertation, we systematically study the security and reliability issues of deep learning models under the following scenarios:

- The size of training data is not large enough to make the model learn statistical features of $p(\mathbf{x})$ precisely. On the one hand, the sample size (training data size) is not large enough to represent $p(\mathbf{x})$. The model may learn the features of $p(\mathbf{x})$ inadequately. On the other hand, the lack of training data may lead to overfitting.
- The testing data are statistically dissimilar to the training data. For example, a deep learning model designed for digit recognition could not provide reliable results if the testing data is an animal image.
- Deep learning algorithms can be easily fooled when the input data are falsified. For example, autonomous vehicles could make dangerous driving plans if one or more sensors are under attack.

1.2 Target Applications

In this dissertation, we concentrate on enhancing the security and reliability of deep learning-based algorithms in processing wireless communications applications data and autonomous driving sensory data. We choose these two application domains due to the following reasons. First, applying deep learning to wireless communications applications is emerging as a promising topic in recent years. It has been shown that deep learning has great potential in addressing some of the most challenging problems in wireless communications and networking, such as modulation recognition [71], transmitter classification [65], spectrum sensing [51], etc. We hope to provide guidelines to help researchers build more reliable and practical deep learning-based wireless communication systems. Second, as one of the most promising application scenarios of deep learning, we anticipate that autonomous driving will play an important role in the near future. Due to the great advances in artificial intelligence (AI), autonomous driving achieved great success in recent years. Automakers such as Waymo

have already shown off their autonomous vehicle technology by providing self-driving taxi service [104]. Prior to the prevalence of autonomous vehicles, it is essential to consider the potential security risks induced by deep learning algorithms.

1.2.1 Wireless Communications Applications

In our research work, we consider the deep learning-based models in the context of two wireless communications applications: (1) transmitter classification, and (2) modulation recognition. Transmitter classification [65] is the problem of classifying transmitters by identifying unique characteristics in the received signal caused by intrinsic attributes of each transmitter. It serves an important role in the assessment of radio frequency (RF) spectrum utilization [77] and identification of bad actors in spectrum sharing ecosystems. For example, in a spectrum sharing system, transmitter classification can be used to quickly identify secondary users that violate spectrum sharing rules and cause harmful interference to primary users [21]. Modulation recognition deals primarily with identifying modulation schemes based on the constellation points. Since conventional modulation recognition approaches were reported to have poor versatility and highly complex [105], automatic modulation recognition has become very useful for receivers to rapidly discriminate signal types.

1.2.2 Autonomous Driving

An autonomous vehicle is capable of sensing its environment and moving safely with little or no human input. The society of automotive engineers (SAE) defines six levels of driving automation, ranked from Level 0—no automation, to Level 5—fully automated [16]. In Level 4 and Level 5, no human control is required to perform the entire driving route. The drivers' attention is likely to be focused on other subjects while the vehicle is moving so

that a significant amount of time is likely to pass before drivers can take proper actions to re-control the vehicle. Therefore driver safety relies purely on the onboard computing systems, which in turn depends on the ability of the autonomous driving system to perceive its surrounding environment. It is of crucial importance to detect any attacks targeting sensors. These sensor attacks may affect driving decisions indirectly by fooling the signal processing algorithms. This could make the autonomous system ignore existent objects or mistakenly detect nonexistent objects, which potentially cause serious accidents.

1.3 Contribution

In this dissertation, we focus on studying and solving the security and reliability issues described in Chapter 1.1.2. In particular, we seek to answer the following questions pertaining to the security and reliability of deep learning models:

- How to analyze the relationship between model performance and training data size quantitatively? How does the model performance change if there exists bias as more training data is added (e.g., unbalanced training data in classification)?

- How the model reliability could be decreased when the testing data are significantly different from training data? How to detect such abnormal testing data effectively?

- How the sensor attacks targeting LIDAR and camera in autonomous driving may impact the perception results? How to detect sensor attacks towards LIDAR and camera effectively without introducing redundant sensors?

The main contributions of this dissertation are summarized below:

1.3.1 Studying how Model Reliability Changes with Sample Size

Compared with other research areas such as image processing and text mining, one key challenge in conducting deep learning-based wireless communication experiments is the lack of training data. In image processing and text mining, there are a plethora of publicly available benchmark datasets. Nevertheless, only a handful of datasets are available for wireless systems or networking research. What's worse, each of those datasets can be used only for specific applications. For example, the *radioml* dataset [71] can only be utilized for modulation recognition research, and cannot be used for other purposes. Therefore, in most cases, wireless communication researchers have to generate their own datasets to train the deep learning models. Because generating datasets is a costly and time-consuming task, it is important to estimate the amount of training data that is required to achieve a target performance, which is referred to as the *sample complexity* problem.

Prior works have demonstrated that testing accuracy can be estimated empirically given the training data size in deep learning-based classification algorithms [14, 36, 70]. However, testing accuracy is not only determined by the model performance but also determined by the size of the testing data for each class. The testing accuracy may vary significantly if a deep learning model is evaluated by different datasets. As a consequence, it is improper and inadequate to only use testing accuracy to describe the model performance. We need to use performance metrics that are determined by the essential attributes of the model, which we refer to as *model-intrinsic* metrics, to evaluate the model performance.

In the first work, we systematically study the relationship between the model performance and the size of training data. More specifically, we use statistical models to describe how model-intrinsic metrics changes with the training data size quantitatively. Our findings and insights are based on numerous results in our transmitter classification and modulation

recognition experiments. In addition, we demonstrate how to estimate the training sample size that is required to achieve a certain performance target and shed light on the trade-off between model performance and the size of the training data.

1.3.2 Detecting Out-of-Distribution Testing Data

In deep learning, it is implicitly assumed that the testing data are sampled from the same distribution as the training data. Based on this assumption, deep learning models trained with the training data can generalize well to unseen testing data. Nevertheless, when deploying deep learning products in real-world applications, there is often very little control over the testing data. Some testing data may be significantly different from the training data. Such a phenomenon is very common in wireless communications applications since the wireless channel environment is highly dynamic. In the testing stage, wireless signals are likely to be transmitted under a channel condition that is significantly different from the channels used for collecting the training data.

From a statistical point of view, data sampled from the same distribution as training data are denoted as *in-distribution* (ID) data. Data sampled from a *significantly different* distribution from the training data are denoted as *out-of-distribution* (OOD) data. The behavior of a deep learning model on OOD data is highly unpredictable. In other words, the existence of OOD data could decrease the model reliability significantly. Besides, attackers can generate fake and irrelevant input testing data intentionally in order to fool a deep learning model. Failing to detect OOD data may expose severe security risks.

In the second work, we first discuss the impact of OOD data in wireless transmitter classification and modulation recognition. Then we propose a new deep learning model called FOOD (**F**eature representation for **OOD** detection), a unified deep learning model that is

able to detect OOD testing data and perform classification tasks simultaneously.

1.3.3 Enhancing Security and Safety in Autonomous Driving

In order to perceive driving conditions, an autonomous driving system needs to carry out the following two steps: sensing and analyzing. Autonomous vehicles first sense their environment using multiple types of sensors such as camera, LIDAR, radar, sonar, etc. Then the autonomous system analyzes the sensory data to extract useful information and the information is used to navigate the vehicle using AI algorithms. Therefore, autonomous vehicles can operate properly and safely only if the following conditions are met: (1) sensors correctly capture stimuli from the environment in their sensory data; and (2) data processing algorithms process the sensory data free of errors to perceive the environment correctly. Hence, ensuring the trustworthiness of the sensor data is crucial for safety.

Unfortunately, vehicle sensors are vulnerable to all kinds of attacks and mishaps. Adversaries can either hack sensors remotely or simply taint or damage sensors physically. In addition, sensors may deviate their calibrated positions due to attacks and mishaps. Such attacks and mishaps indirectly impact driving decisions by making sensors fail to correctly perceive the surrounding driving environment. *Perception errors* can be caused by attacks, malfunctioning sensors, or unintentional mishaps. In particular, we focus on perception errors induced by attacks and coin the term *Perception Error Attacks* (PEAs) to denote such attacks.

In the last work, we discuss the impact of PEAs on autonomous vehicles equipped with LIDAR and stereo cameras and propose a countermeasure called LIFE (**L**IDAR and **I**mage data **F**usion for detecting perception **E**rrors) [54]. LIFE detects PEAs by analyzing the consistency among different types of sensory data without requiring additional hardware. LIFE is able to detect perception errors, irrelevant of whether they are due to PEAs or

faulty sensors. The performance of LIFE has been evaluated extensively using the KITTI dataset, the most widely used autonomous driving benchmark suite [28].

1.4 Organization

The remainder of this dissertation is organized as follows:

- Chapter 2 introduces the machine learning and deep learning technical backgrounds that we use throughout this dissertation.
- Chapter 3 explains the general procedure of transmitter classification and modulation recognition.
- Chapter 4 systematically studies the relationship between model performance and training data size in wireless communications applications.
- Chapter 5 describes how OOD data impact the model reliability and explains how FOOD is able to detect OOD testing data while maintaining the same accuracy with standard CNN-based classification algorithms.
- Chapter 6 demonstrates how PEAs may impact the security and safety in autonomous driving and how LIFE detects PEAs effectively.
- Chapter 7 concludes this dissertation and points out future research directions.

Chapter 2

Technical Background

In this chapter, we introduce some essential machine learning and deep learning technical backgrounds that are used throughout this dissertation. In Chapter 2.1, we introduce neural networks, the fundamental structure of deep learning models. We discuss the training process of deep learning models in Chapter 2.2. Then we explain details about variational autoencoder (VAE)[1], a generative model that has a great potential in detecting OOD data in Chapter 2.3. Chapter 2.4 describes the details of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [23], which is beneficial in processing LIDAR data in autonomous driving systems. Finally, we explain camera models in Chapter 2.5.

2.1 Neural Network

Neural networks are computing models inspired by the biological neural networks that constitute animal brains. They serve as the basic structures of deep learning models. A neural network consists of a collection of connected nodes called neurons. In theory, a three-layer neural network can uniformly approximate any continuous function to arbitrary accuracy given enough hidden nodes. It is served as the theoretical foundation of deep learning.

An artificial neural network (ANN) is composed of an input layer, one or more hidden layers, and one output layer. It is also called deep neural network (DNN) when the number of hidden layers is large. In each hidden layer or output layer, all neurons first compute

the weighted sum of values in the connected neurons from the previous layer, then apply activation functions to the weighted sum and forward the results to the connected neurons in the next layer. In general, the activation functions are nonlinear. The activation functions used in this dissertation are summarized below:

- $\text{ReLU}(x) = \max\{0, x\}$. It is widely used in the hidden layers.
- Leaky $\text{ReLU}(x) = \max\{\alpha x, x\}$, where $\alpha \in (0, 1)$ is a predefined value. If $\alpha = 0$, Leaky ReLU degenerates to ReLU. Compare to ReLU, leaky ReLU does not have the gradient vanishing issue when $x < 0$.
- $\text{Sigmoid}(x) = 1/(1 + \exp(-x))$. It is commonly used in the output layer if the outputs are a series of values range in $[0, 1]$.
- Softmax function. It is the most widely used activation function in the output layer in deep learning-based classification algorithms. The input of a softmax function is a vector $\mathbf{z} = (z_1, \cdots, z_n)$, and the output is a vector $\mathbf{o}$ with $o_i = \exp(z_i)/\sum_{m=1}^{n} \exp(z_m)$. The value of o_i represents the probability that the input belongs to class i. The classification result is chosen to be $\text{argmax}_i\, o_i$.
- $\text{Softplus}(x) = \ln(1 + \exp(x))$. It maps any real number to a positive number.

Convolutional neural network (CNN) is a type of DNN that is beneficial in processing data with strong spatial correlation (e.g., images). The input of a CNN is a 3D tensor (Height×Width×Channel). After applying a series of convolutional layers and pooling layers alternatively, the output tensor is flattened to form a one-dimensional vector. The flattened vector is then fully connected to one or more hidden layers. The final hidden layer is connected to the output layer.

Recurrent neural network (RNN) is a type of DNN that is beneficial in processing data with strong sequential correlation (e.g., natural language and time series). The neurons in each hidden layer are updated by both the current external input and the hidden layer one step back in time. In this manner, the hidden layers are able to store information about past states. The impact of inputs at a certain time becomes less as the states evolved with time.

RNN and CNN can be combined to process video frames. The CNN is used to extract essential features from images and reconstruct original images from the features. The RNN is used for processing the extracted image features sequentially. In Chapter 6, we will explain how to predict the next autonomous driving camera image in a video stream based on images from previous frames.

2.2 Stochastic Training Process

Machine learning models are trained by minimizing the loss function associated with the training data. The most common approach to solve the optimization problem is gradient descent, a first-order iterative algorithm to find a local minimum of a differentiable function. However, it needs to run through all the samples in the training data set to perform a single update for a parameter in each iteration. As a consequence, it takes a very long time to find a local minimum when the size of training data is large. To overcome this issue, researchers usually apply the stochastic gradient descent (SGD) method to train deep learning models. Different from the gradient descent method, SGD uses only one sample from the training data set to update parameters in each iteration, which makes SGD much faster than the gradient descent method.

Although SGD is able to reduce the training time significantly, it generates high variances of the results. This is because the model parameters are updated based on a single data

point in each iteration, which may not necessarily reduce the loss of all the training data. Mini-batch gradient descent is a variation of SGD that uses a batch of training data to update model parameters in each iteration. It reduces the variances in the training process while maintaining a high training speed. In our research works, we choose the mini-batch gradient descent method to train our deep learning models.

But for loss function with a high condition number (ratio of the largest to the smallest singular value of the Hessian matrix), setting a single learning rate for all parameters may cause the model to either make slow updates along shallow directions or make jitter updates along steep directions. We choose to use adaptive learning rate (e.g., RMSProp) to overcome this issue. Moreover, we use momentum to increase the chances to find better local minimums. In our research works, we choose to use Adam [41], which combines RMSProp and momentum, as our training optimizer.

We apply batch normalization [38] to further reduce the training time. In machine learning, it is common to normalize the data before passing the data to the input layer. Batch normalization borrows the idea of data normalization and normalizes the input values in each hidden layer. According to our experiments, applying batch normalization can reduce the training time significantly.

2.3 Variational Autoencoder

FOOD is a model to detect OOD testing data in classification algorithms. The architecture of FOOD is based on VAE. FOOD extends the capability of detecting OOD data to classification by modifying the framework of vanilla VAE.

Before explaining the concept and technical details of VAE, it is necessary to introduce

autoencoder first. An autoencoder is composed of an encoder and a decoder. The encoder takes in a high-dimensional input $\mathbf{x}$ and converts it into a representation $\mathbf{z}$ with much lower dimensionality than $\mathbf{x}$. Then the decoder can convert $\mathbf{z}$ back to $\mathbf{x}$ with high fidelity. In this case, autoencoder can be regarded as a dimension reduction method or a compression method. However, an autoencoder is only guaranteed to work well for the training data. The quality of the reconstructed data is unpredictable for any data not used for training. In other words, an autoencoder may not generalize well to data that is not used for training.

VAE can overcome this issue by encoding $\mathbf{x}$ into multiple representations $\mathbf{z}$. A VAE is also composed of an encoder and a decoder. But instead of learning a single representation $\mathbf{z}$, the encoder of a VAE maps an input $\mathbf{x}$ to a probability distribution $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$. The decoder is able to reconstruct $\mathbf{x}$ with high quality from random variables $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$.

In a standard VAE, the encoder actually maps the high-dimensional input data $\mathbf{x}$ into two vectors $\mu(\mathbf{x})$ and $\Sigma(\mathbf{x})$ of the same dimension. The probability distribution is chosen such that $\boldsymbol{\mu}_x = \mu(\mathbf{x})$ and $\boldsymbol{\Sigma}_x$ is a diagonal matrix whose diagonal entries equal to $\Sigma(\mathbf{x})$. Without regularizations, the encoder can either return distributions with tiny variances or return distributions with very different means. In such cases, a VAE degenerates to an autoencoder. Hence, VAE encourages different distributions $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ to "overlap" with each other as much as possible. To satisfy such requirements, the loss function of VAE is a weighted sum of reconstruction errors and the Kullback–Leibler (KL) divergence between $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ and the standard multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

To train the encoder and the decoder in VAE, we need to backpropagate through random samples $\mathbf{z}$. This causes a serious problem because backpropagation cannot flow through random variables. To overcome this obstacle, we use the reparameterization trick [42]. Instead of sampling $\mathbf{z}$ from $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ directly, we rewrite $\mathbf{z}$ as $\mathbf{z} = \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_x^{1/2}\epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Now the sample $\mathbf{z}$ can be considered as a function taking parameters $\epsilon, \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x$, where $\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x$

come from the encoder. This allows $\boldsymbol{\mu}_x$ and $\boldsymbol{\Sigma}_x$ to remain the trainable parameters of the model while still maintaining the stochastic property of the entire model via ϵ.

2.4 DBSCAN

Clustering is the task of grouping a set of data points in such a way that objects in the same cluster are more similar to each other than to those in other clusters. It is one type of unsupervised learning, and the performance evaluation is very subjective. It can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to find them efficiently.

We use DBSCAN to process 3D LIDAR data points in LIFE. The high-level idea of DBSCAN is to group data points in high-density regions while mark as outliers when data points located in low-density regions. Before explaining details for the DBSCAN algorithm, we need to introduce the following definitions:

- Two points are *neighbors* of each other if their distance is no larger than ϵ. The most commonly used distance is Euclidean distance.
- A point is a *core point* if it has more than MinPts neighbors. MinPts is a parameter specified in advance.
- A *border point* has fewer than MinPts neighbors within ϵ, but is a neighborhood of a core point.
- A *noise point* is any point that is neither a core point nor a border point.

Figure 2.1 explains these terms in a more intuitive manner. The algorithm of DBSCAN is described as follows:

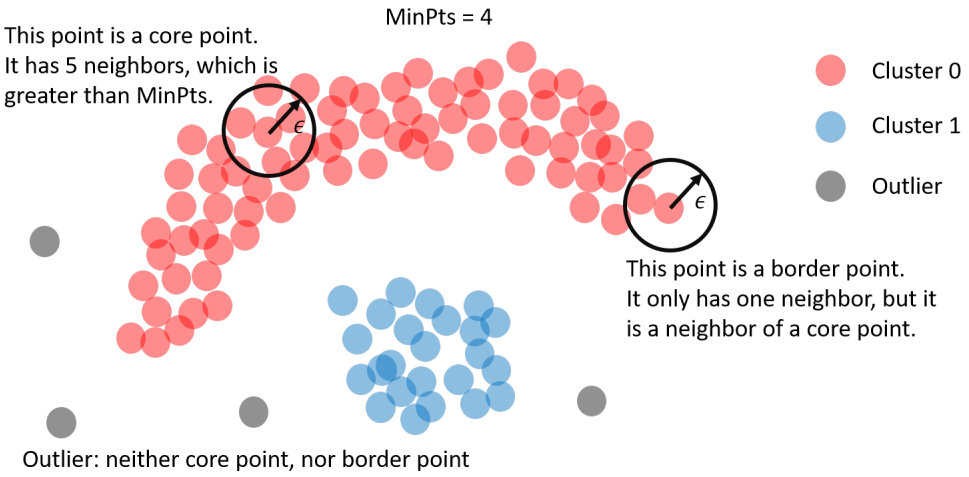


Figure 2.1: Illustration of DBSCAN.

1. Randomly pick a point that has not been assigned to a cluster or has been designated as a noise point. Compute the number of neighbors to determine if it is a core point. If yes, start a cluster around this point. If no, label the point as a noise point.

2. Once we find a core point, expand the cluster by adding all neighbor points within ϵ to the cluster. If a noise point is added, change its status from noise to border point.

3. Repeat step 1 and step 2 until all points are either assigned to a cluster or designated as a noise point.

Compared with other clustering algorithms, DBSCAN has the following advantages. First, DBSCAN can determine the number of clusters automatically. Most of the other clustering algorithms such as K-means and hierarchical clustering require users to determine the number of clusters in advance, which is impossible when processing LIDAR data. Second, DBSCAN makes no prior assumptions on the statistical distribution of data. DBSCAN can cluster data points into arbitrary shapes. Several clustering methods like K-means can only apply to data for which a centroid is meaningful. Finally, DBSCAN can handle outliers effectively. The existence of noisy data won't impact the performance of DBSCAN.

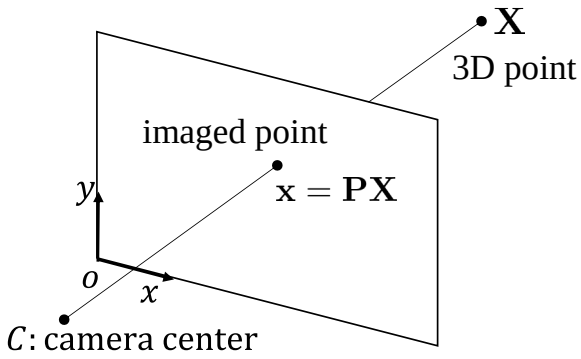


Figure 2.2: Camera model diagram.

2.5 Camera Model

The (projective) camera model carries the 3D to 2D mapping using *homogeneous coordinates* instead of Cartesian coordinates. Given a point (x, y) in a $\mathbb{R}^2$ Cartesian coordinate, the triple (kx, ky, k) for any $k \neq 0$ is called a homogeneous coordinate for this point. A point $\mathbf{x} = (x, y)$ lies on a line $\mathbf{l} = (a, b, c)$ if and only if $ax + by + c = 0$, which implies $(kx, ky, k)(a, b, c)^{\mathrm{T}} = 0$. So homogeneous coordinates will be invariant to scaling in describing lines in $\mathbb{R}^2$ and planes in $\mathbb{R}^3$. The original Cartesian coordinate is recovered by dividing the first two positions by the third. For example, Cartesian point $(2, 3)$ can be represented as $(2, 3, 1)$ and $(4, 6, 2)$ in homogeneous coordinates. The same definition applies to points in $\mathbb{R}^3$ Cartesian coordinate. *In the rest of this dissertation, we will use homogeneous coordinates to represent points in* $\mathbb{R}^2$ *and* $\mathbb{R}^3$*, unless we specify it is Cartesian coordinate.*

Suppose $\mathbf{X}$ is a point in a 3D coordinate system, then the imaged point $\mathbf{x}$ is the intersection between the ray passing through camera center C and point $\mathbf{X}$ and the camera image plane, as illustrated in Figure 2.2. Such mapping is characterized by a 3×4 *calibration matrix* $\mathbf{P}$ as $\mathbf{x} = \mathbf{PX}$. In practice, $\mathbf{P}$ can be computed with very high precision using multiple view geometry techniques [33]. In our research work, we assume that errors only exist in the measured image coordinates ($\mathbf{x}_i$), not in calibration matrices ($\mathbf{P}$).

Chapter 3

Classification Problems in Wireless Communications Applications

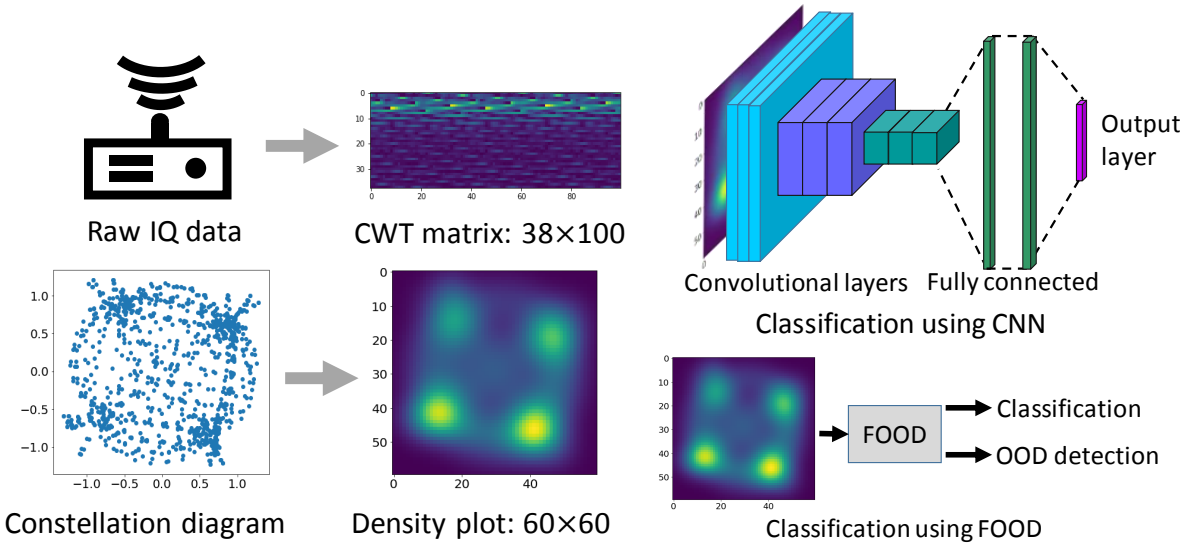
3.1 Transmitter Classification

Many components of hardware devices exhibit some attributes unique to each device. For example, the power amplifiers used in many wireless devices tend to exhibit non-linearities when they are operated at high power [81]. These attributes can be extracted from RF signals and are very useful in transmitter classification.

In our transmitter classification experiments, we generated our own dataset using software-defined radios. Transmitter classification consists of three steps: (1) data collection, which involves extracting raw IQ data from the received signals, (2) signal processing, which involves extracting important features to help classify the transmitters, and (3) classification using CNN models based on the processed data.

3.1.1 Data Collection

The data collection consists of all the processes involved in the waveform generation, transmission, and reception of the signals. All the transmissions were done over the air with a fixed position of the receiver and varying positions of the transmitters.



(a) Signal processing. (b) Classification.

Figure 3.1: Procedure of transmitter classification and modulation recognition.

We utilized the GNU Radio Companion signal processing blocks to generate OFDM packets. The VERT 2450 omnidirectional antennas were used to transmit and receive the signals. A random source that generates a stream of bits is created. These bits are then mapped to an OFDM waveform using QPSK modulation with an FFT length of 512 and occupy 200 tones with a cyclic prefix of 128. A bandwidth of 5 MHz was used for the transmission. At the transmission stage, the USRP hardware driver links GNU radio with the transmitter hardware. For transmission, the OFDM waveform is up-converted to a center frequency of 2.45 GHz. For reception, we assume that the receiver knows the center frequency and bandwidth of the transmitter. The receiver samples the incoming signals at a 10 MHz sampling rate at the same center frequency as the transmitter. We use the UHD USRP Source block in GNU radio to down-convert the signal to the baseband frequency. The received signal is sent to a low noise amplifier, after which the signal is split into the in-phase (I) and quadrature (Q) components at the baseband. The raw IQ data are streamed to a host computer through the host interface. Finally, the IQ data are sent to the continuous

wavelet transformation (CWT) processing step.

3.1.2 Data Processing

In our transmitter classification scheme, the classification is based on 100 continuous IQ data. Although we can take the raw IQ data as input and let the deep learning algorithms learn the inherent features by themselves, in practice, it requires far more training data and is less effective than applying appropriate signal processing techniques. In this paper, we chose CWT to process the 2×100 raw input IQ data. After CWT, a matrix of dimension 38×100 is produced.

CWT is a time-frequency analysis tool that is used to process the raw IQ data to enable a clear depiction of the frequency components of the signals. The transform introduces spatial correlations that are much more difficult to observe in the original signals. CWT helps to magnify the features in the received signal for classification. It does this by decomposing the signal into wavelets using basis functions, which are scaled and shifted versions of the mother wavelet [59, 82]. Given the different available CWT basis functions, the Morlet wavelet was selected because of its wide usage and ease of implementation. A two-dimensional matrix where the rows correspond to the scales and the column corresponding to the length of the signal [59] is then generated. The mathematical details of choosing the scales and computing the CWT matrix elements can be found in [82]. *In the rest of this paper, we will refer to this* 38×100 *CWT matrix as a data point in our transmitter classification experiments.*

3.1.3 Classification

An example of a CWT matrix is illustrated in Figure 3.1. The color in the CWT matrix represents the relative values among different matrix entries. We can observe that a CWT

matrix has a very strong spatial correlation. In Chapter 4, we apply standard CNN-based deep learning algorithms to perform the classifications. The input to the CNN classifier is a 38×100 CWT matrix, and the output of the classifier is a vector of non-negative values that add up to one, representing the probability that the input belongs to each class. The predicted class is the one associated with the maximum probability value in the output. In Chapter 5, we propose a new classification method in FOOD.

3.2 Modulation Recognition

We use the RF dataset provided by *DeepSig* [72] to conduct the modulation recognition experiments. It contains both synthetic channel impairments and over-the-air recordings of several modulation schemes. Data are collected under varying SNR levels from -20 dB to +30 dB with an increment of 2 dB. There are 4096 data at each SNR value for each modulation scheme. Each data point in the dataset is a 1024×2 array representing 1024 constellation points in the constellation diagram. The objective of modulation recognition is to infer the received signal modulation scheme based on the IQ components of the 1024 constellation points.

3.2.1 Data Processing

Some prior studies directly take the IQ components of constellation points as inputs of classification algorithms. However, given an array of constellation points, the corresponding constellation diagram is irrelevant to the order (or arrangement) of the individual constellation points. Applying convolutional filters to the 1024×2 array may learn features that are related to the order of the constellation points and cause overfitting issues. To overcome

these issues, we construct density plots [89] based on the constellation points, and use these plots as inputs to the classifiers. An example of a density plot is illustrated in Figure 3.1. The constellation points can be considered as samples of a probability distribution. Each pixel in a density plot represents the estimated probability density function (PDF) of a random variable. Since there exist some outliers with extremely large values of I or Q components, we first select constellation points that have the values of both I and Q components within $[-1.5, 1.5]$, where more than 99% of the constellation points are located within this range. Then we divide the interval $[-1.5, 1.5]$ into 60 bins for both axes. Finally, we estimate the PDF of each cell using the method provided in [89] and obtain *a density plot of size* 60×60*, which we refer to as a data point in our modulation recognition experiments.*

3.2.2 Classification

Since each pixel in a density plot represents an estimation of the PDF of the constellation points, the pixels have a very strong spatial correlation. Similar to transmitter classification, we apply standard CNN-based algorithms to perform modulation recognition in Chapter 4. The input to the CNN classifier is the 60×60 density plot, and the output of the classifier is a label associated with the modulation schemes. In Chapter 5, the classification is based on our newly proposed method. In our modulation recognition experiments, we used the same deep learning model architectures that were used in the transmitter classification experiments. They only differ in the number of neurons in the last convolutional layer.

Chapter 4

Relationship between Model Performance and Sample Size

4.1 Introduction

Unlike in other deep learning application domains such as image processing and text mining where there is a plethora of publicly available benchmark datasets, only a handful of datasets are available for wireless systems or networking research. Moreover, it is extremely difficult, if not impossible, to create a dataset that can be used for training/testing different models to address different problems in various wireless communications/networking applications. Hence, it is not surprising that there are no publicly available benchmark datasets for the vast majority of wireless research problems. Researchers working on these problems need to generate their own datasets, which is very resource intensive and time consuming. Hence, there is a critical need to understand the relationship between the dataset size and the corresponding performance of the deep learning model. This is especially important in the application of deep learning to wireless communications/networking problems.

It is well known that increasing the training data size improves the performance of a deep learning model. However, researchers have yet to achieve a systematic understanding between the data size and model performance. Although several prior studies [46, 62, 87] have explored the utilization of deep learning for addressing problems in wireless communications

and networking, these works focused only on the feasibility or the advantages of applying deep learning to those problems.

Previous works have demonstrated that in deep learning-based classification algorithms, the testing accuracy can be estimated empirically given the training dataset size [14, 36, 70]. However, testing accuracy is related to the size of the testing data in each class. Since a model has different capabilities in predicting data from different classes correctly, testing accuracy may vary significantly if the model is evaluated with another dataset that has a different proportion of data points for each class. For instance, suppose we have a classifier that trivially classifies all animal images as cats. Then testing accuracy solely depends on the fraction of cat images in the testing dataset. When deploying deep learning products to the real world, different users may use the same product under completely different scenarios. It is likely that these users do not have prior knowledge on how many data points in each class will be used for testing, hence, testing accuracy alone does not paint a complete picture of the model performance. We need to use performance metrics that are determined by the essential attributes of the model, which we refer to as *model-intrinsic* metrics, to evaluate the model performance. We will explain in section 4.3 that in a multi-class classification algorithm, the performance metric *recall for each class* is model-intrinsic.

In this work, we systematically study the sample complexity problem associated with recall for each class. The sample complexity of a machine learning algorithm is defined as the number of training samples that are required to achieve a performance target with high probability [4]. In practice, acquiring large amounts of training data to train deep learning models can be expensive in terms of man-hours, equipment running time, cost, etc. On some occasions, with additional tens of thousands of training data, only a very slight performance improvement can be observed. This makes it necessary to investigate how scaling training dataset size impacts performance, which is important when researchers need to generate

their own datasets.

To the best of our knowledge, this work represents the first-ever study on the sample complexity problem in the context of model-intrinsic metrics and the reliable estimation of those metrics using statistical models. Although the sample complexity problem has been thoroughly studied in non-deep learning models, such as logistic regression and linear regression [5, 26], it has not attracted much attention from the deep learning research community. Theoretical analysis requires analytical formulas related to the models, which is intractable for deep learning due to the huge number of parameters. Sample complexity in deep learning was studied empirically in [36, 70], but the conclusions are constrained to fixed testing datasets. Besides, [36, 70] assume that all the training data are sampled uniformly and independently from the overall training dataset when they increase the training dataset size. Those works did not provide insights into how changing the proportion of each class's training data when more training data is added can impact model performance.

The main contributions of this work are summarized below:

- We systematically study the sample complexity problem associated with recall for each class, a model-intrinsic performance metric that can be used to accurately quantify the relationship between model performance and the size of training data. Our findings are based on data collected from numerous experiments conducted with software-defined radios and public RF datasets.
- We propose modified definitions of learning curves to help researchers understand the sample complexity problem in a more clear manner.
- We demonstrate how to estimate the size of the training data that is required to achieve a certain performance target and shed light on the trade-off between model performance and the size of the training data.

4.2 Related Work

4.2.1 Deep Learning in Wireless Communications Applications

Previous studies have demonstrated that deep learning algorithms can automatically extract device-specific features by analyzing a huge amount of training data, hence, improving the performance of transmitter classification. For instance, Merchant *et al.* [62] utilized deep learning to fingerprint IEEE 802.15.4 devices. In addition, Youssef *et al.* [109] investigated several machine learning algorithms with wavelet transform to identify and classify wireless transmitters.

Automatic modulation recognition has attracted much attention in recent years. For example, [73, 78] used CNN models to recognize modulation schemes based on sequences of constellation points. Several other papers [40, 100, 105] discuss the challenges of the application of deep learning to modulation recognition in wireless communication.

4.2.2 Sample Complexity

For non-deep learning models such as linear regression and logistic regression, sample complexity can be estimated analytically using Vapnik-Chervonenkis (VC) dimensions [98], a measure of the capability of a classification model. The discrepancy between training and generalization error is upper-bounded by a quantity that grows as the model's VC dimension grows and shrinks as the number of training examples increases. However, we cannot use the VC dimensions to analyze sample complexity in deep learning algorithms because they cannot be solved analytically. Moreover, VC dimension is used for estimating the full capacity of a model. A deep neural network generalizes incredibly well by exploring only a small portion of its capability. Analyzing its full capacity is not useful for practical applications.

The relationship between training dataset size and model accuracy was empirically investigated in [36, 70]. One take-away from their work is that testing error can be modeled using inverse power-law. Likewise, in the medical bio-informatics field, authors in [14] studied the optimum size of training data for optimal performance for medical image classification.

Sample complexity in deep learning was theoretically analyzed in [20]. However, a few impractical assumptions, such as only using linear activation functions in each layer, were made in [20] to simplify the mathematical formulation, which limits the utility of the findings. A non-vacuous generalization bound was studied in [22]. Unfortunately, the provided bounds are still very loose in practical applications.

4.3 Model-Intrinsic Metrics in Classification

In section 4.1, we explained that testing accuracy is not a model-intrinsic metric. Similarly, precision and F1 score are not model-intrinsic as well. In this section, we first explain why the recall for each class is a model-intrinsic metric. Next, we group classification errors into two categories and discuss how they change as more training data is added. Finally, we discuss certain conditions for this model-intrinsic metric to be estimated reliably using statistical models.

In the rest of this chapter, we use the following notations:

- $\mathcal{X}$: sample space of the input data.
- $\mathbf{x}$: a data point sampled from $\mathcal{X}$.
- y: the label of $\mathbf{x}$.
- M: number of classes.

- $\mathcal{Y} = \{1, 2, \cdots, M\}$: the set of class labels.
- p_i: the true distribution of $\mathbf{x}$ that belong to class i.
- A_i: a subset of $\mathcal{X}$ computed by the classifier such that all $\mathbf{x} \in A_i$ are predicted to belong to class i.

In classification algorithms, it is assumed that each training data and testing data that belong to class i are sampled independently from the same distribution p_i. Based on this assumption, deep learning models trained with the training dataset can generalize well to unseen testing data. In practice, the distributions $p_i, i \in \mathcal{Y}$ are unknown and cannot be derived analytically in high-dimensional spaces. A classifier extracts features that are highly correlated with p_i from a finite amount of training data and aggregates these features to predict the labels of the testing data. A classification algorithm separates the entire sample space $\mathcal{X}$ into disjoint sets $A_i, i \in \mathcal{Y}$, where $\forall \mathbf{x} \in A_i$ are predicted to be class i. The probability of predicting a class i data point as class j is:

$$\int_{\mathbf{x} \in \mathcal{X}} p_i(\mathbf{x}) \mathbf{1}_{\mathbf{x} \in A_j} d\mathbf{x} \tag{4.1}$$

where $\mathbf{1}_{\mathbf{x} \in A_j}$ is equal to 1 if $\mathbf{x} \in A_j$ and is equal to 0 if $\mathbf{x} \notin A_j$. Based on the law of large numbers, for large size of testing data, the value of eq. (4.1) is close to the fraction of class i data that is predicted to belong to class j. When $j = i$, eq. (4.1) represents the fraction of class i data points that can be predicted correctly, which is *recall for class i*. The definition of recall for a class should not be confused with the definition of recall in the whole testing dataset, which is the sum of true positives across all classes divided by the sum of true positives and false negatives across all classes. *When we use the term recall in this chapter, we are referring to recall for certain classes rather than recall for the whole testing dataset.* Since p_i is a fixed function and A_i is solely determined by the deep learning model, the value

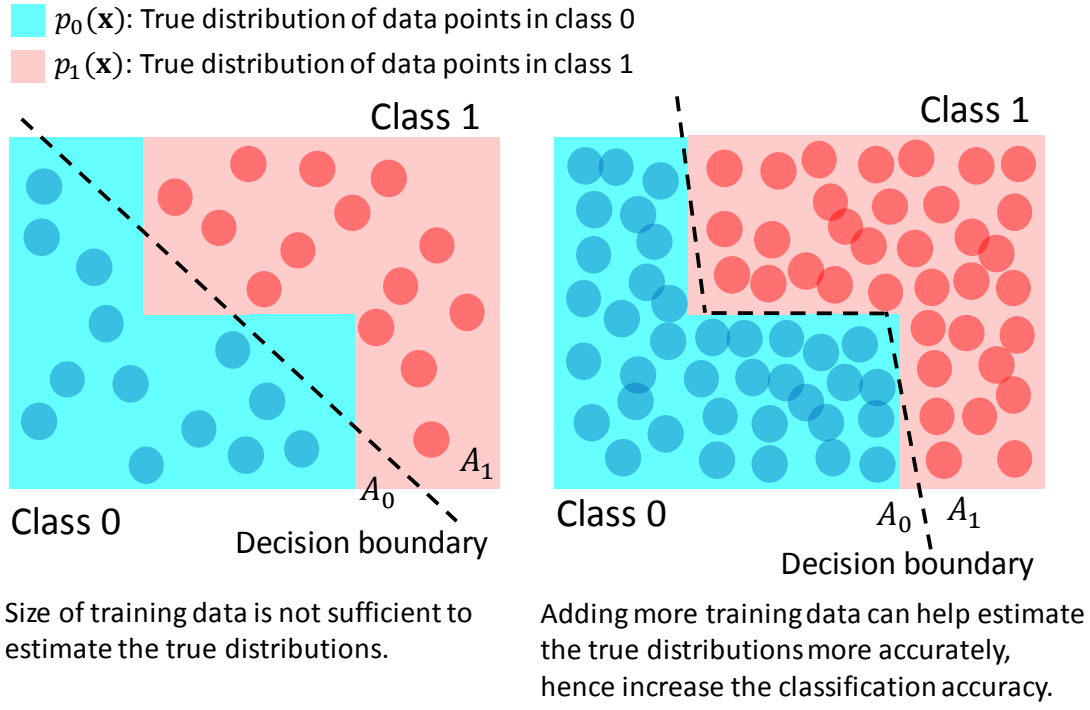
of eq. (4.1) is model-intrinsic and is irrelevant to the size of testing data for each class.

In this chapter, we analyze *how the recall for each class changes as more training data is added*. Before discussing more details, we need to understand the sources of classification errors. We categorize classification errors into *reducible classification errors* and *irreducible classification errors*, as illustrated in Figure 4.1:

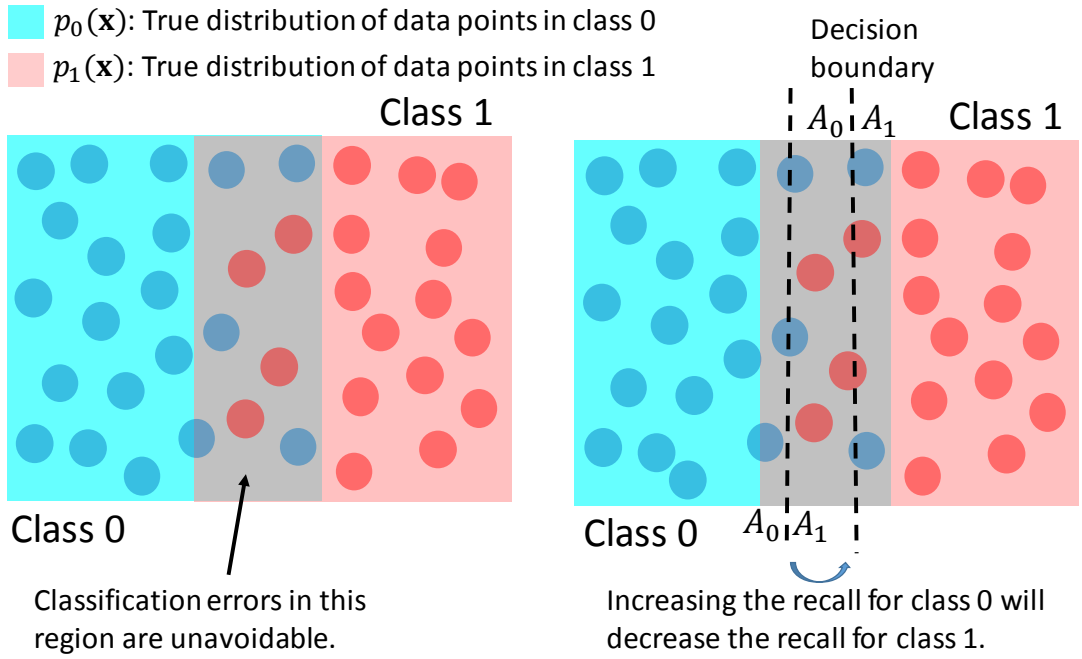
(1) Reducible classification errors: some classification errors exist because the size of the training data is not sufficient to estimate p_i precisely. The sets A_i derived from classification algorithms are different from the optimal solution derived from the true distributions p_i. *Such classification errors can be reduced by adding more training data*, as shown in Figure 4.1a.

(2) Irreducible classification errors: there may exist classification errors that cannot be reduced even when the model is trained with a sufficient amount of data. One reason is that the capacity of a model is not high enough to accommodate all the essential statistical features from the training data. In particular, a neural network without a sufficient number of layers and neurons may fail to extract enough useful features for classification. The other reason is that there exist some data points $\mathbf{x}$ such that the value of $p_i(\mathbf{x})$ is high for multiple $i \in \mathcal{Y}$, which we refer to as *ambiguous data points*. For example, as shown in Figure 4.1b, the data points that belong to class 0 are uniformly distributed in the cyan region, and the data points that belong to class 1 are uniformly distributed in the pink region. There is an overlap between these two regions. All the points in this overlap region are ambiguous data points. No matter how we choose A_i, there always exist classification errors. *Such classification errors are caused by the overlap among distributions p_i. Increasing the recall for one class will decrease the recall for other classes.*

When irreducible classification errors are present, recall for each class depends heavily on how the classifier predicts the ambiguous data points. Since a classifier is trained to minimize



(a) Reducible classification errors due to the lack of training data.



(b) Irreducible classification errors due to the overlap among multiple p_i

Figure 4.1: Two types of classification errors.

the loss function of the training data, the recall for each class is impacted by the number of sampled ambiguous data points in each class, which in turn is impacted by the size of the training data for each class (assume that training data is sampled identically and independently). Although we can decrease the reducible errors by adding more training data, changing the proportion of training data for each class may cause the classifier to predict ambiguous data points in different ways. Therefore, when there exist irreducible classification errors, it is very challenging to estimate the recall for each class correctly if the proportion of the training data size for each class changes unexpectedly.

On the contrary, if there is no irreducible classification error, the recall for class i depends on how the model extracts features based on the samples of p_i. Since there is no overlap among $p_i, \forall i \in \mathcal{Y}$, data points that do not belong to class i have little to no impact on the feature extraction process of class i. Therefore, the recall for class i has almost no correlation with the size of training data in other classes.

In this section, we claim that the recall for class $i, \forall i \in \mathcal{Y}$ follows the extended inverse-power law with the size of training data for class i if either of the following two conditions is satisfied: (1) The fraction of training data for each class (roughly) remains the same as more training data is added to improve model performance, i.e., n_i/n is a constant for any i when n increases, where n_i is the training data size for class i and n is the overall training dataset size. (2) No irreducible classification errors are present using the given deep learning model. On the one hand, we conduct transmitter classification experiments and use a publicly available modulation recognition dataset to justify such conclusions in Chapter 4.7. On the other hand, we demonstrate that the extended inverse-power law model may not apply to the cases if neither of the aforementioned conditions is satisfied. When there exists a large portion of irreducible classification errors, recall for certain classes may even decrease when we add more training data.

In wireless communications applications, it is very common to have irreducible classification errors since radio propagation data is highly sensitive to wireless channel conditions. The prevalence of irreducible classification errors becomes more pronounced as the signal-to-noise ratio (SNR) drops. Hence, a sufficient condition to estimate recall accurately using the extended inverse-power law model in wireless communications applications is to keep the proportion of each class's training data fixed when more training data is added in an attempt to improve a model's performance.

4.4 Experimental Setup

4.4.1 Experimental Setup for Transmitter Classification

We used four USRP 2921 devices and four USRP b200 devices as transmitters to conduct our experiments. Our task is to identify these eight transmitters based on the received IQ data. All the IQ data were collected in a lab environment. We considered the following two transmission scenarios. The diagrams for these scenarios are illustrated in Figure 4.2.

- Short range communication (SRC): the transmitter and the receiver are only 1 meter away without any obstacles between them. The impacts of the channel are minor and the performance of the classification algorithms are expected to be very good.
- Line-of-sight (LOS): the transmitter and the receiver are 1.5 meters above the ground and at a distance of 5.42 meters with a clear line-of-sight channel. There exist irreducible classification errors in this scenario.

For each device, we collected 50K raw IQ data arrays of size 2×100 under each transmission scenario. These raw data were subsequently pre-processed using CWT and then used for

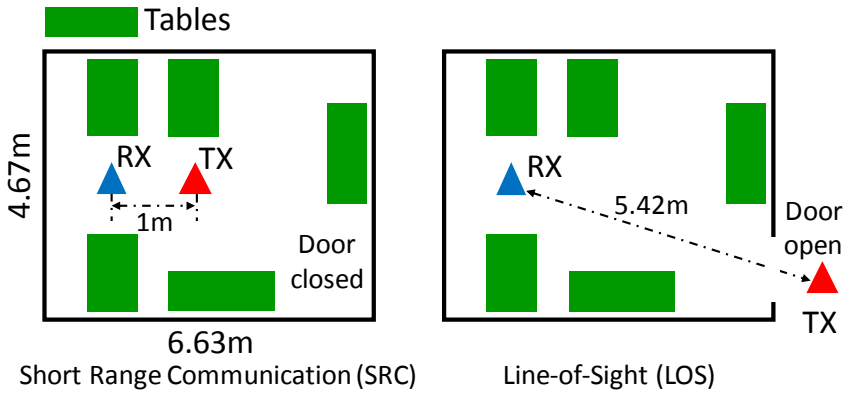


Figure 4.2: Diagrams for different transmission scenarios.

training and classification. We did not consider non-line-of-sight transmission scenarios since such scenarios would result in extremely poor classification performance. Some prior works such as [87] modified the RF block of transmitters to amplify the attributes of the transmitters to facilitate more reliable classification or identification of the transmitters. In our experiments, we did not make such modifications and only relied on the features inherent in the transmitter devices for classification. These features are subtle, and cannot be extracted reliably by a deep learning model when the SNR is too low.

4.4.2 Experimental Setup for Modulation Recognition

We use the public RF dataset [73] to study the sample complexity problem in the context of modulation recognition. Our experiments focus on classifying the following eight digital modulation schemes: BPSK, QPSK, 8 PSK, 8 ASK, 16 QAM, 32 QAM, 32 PSK, and GMSK. We carried out two types of experiments as explained below:

- Classify modulation schemes based on the constellation points when the SNR is between 12 dB and 20 dB. In this scenario, the recall for most modulation schemes is close to 1.0 given a sufficient amount of training data.

- Classify modulation schemes based on the constellation points when the SNR is between 2 dB and 10 dB. In this scenario, there exist irreducible classification errors due to the high noise level. In our experiments, we were not able to perform modulation recognition with reasonable accuracy when the SNR was 0 dB or lower.

4.5 Implementation Details of CNN

In this section, we describe the architecture of the CNN models and the training parameters used in our experiments. To demonstrate that our findings are applicable to a wide range of CNN models, we built three CNN models with significantly different complexities to study the sample complexity problem. They vary in both the number of convolutional layers and the number of channels (neurons) in each layer. Henceforth, we refer to these three CNN models as *Simple CNN*, *Medium CNN*, and *Complex CNN*. The architectures for the three CNNs are summarized in Table 4.1. The Simple, Medium, and Complex CNNs have $152K$, $2.4M$, and $19M$ parameters, respectively.

All the three CNN models are composed of multiple convolutional layers followed by two fully connected layers and one output layer. Leaky ReLU (Leaky ReLU$(x) = \max\{\alpha x, x\}$, $\alpha \in (0, 1)$ is a predefined parameter) activation function with $\alpha = 0.2$ is applied to all convolutional layers and fully connected layers. The softmax function is applied to the output layer. We apply batch normalization [38] to all convolutional layers, but not to the fully connected layers and the output layer. In addition, we take stride = 2 in the convolutional layers for downsampling instead of using 2×2 pooling layers. Such modifications improve performance and reduce the variance of the results in multiple training processes.

The values in the CWT matrix are in the order of $10^{-5} \sim 10^{-4}$, and the values in density plots are in the order of $10^{-3} \sim 10^{-1}$. Input data with such values are not suitable for CNN

training. Therefore, for each input data point $\mathbf{x}$, we normalize $\mathbf{x}$ as $\mathbf{x}' = \mathbf{x}/x_{\max}$, where $x_{\max}$ is the maximum entry value in $\mathbf{x}$. The training optimizer and parameters remain the same for all three CNN models. We use the Adam optimizer with default parameters recommended in [41]. The batch size is 64, and the learning rate is 5×10^{-4}.

All CNN parameters are initialized randomly following truncated Gaussian distribution with a standard deviation of 0.02. Then we randomly select non-repeating 64 training data to update CNN parameters iteratively using gradient descent methods. When all the training data points have been sampled once to update the CNN parameters, we complete one training *epoch*. After training for a certain number of epochs, the training accuracy fluctuates around a certain value. When this is observed, we conclude that the *training process* is completed and, thus, stops updating the parameters.

We are aware that in a typical deep learning training process, we should use a validation dataset to monitor the performance and select the model that achieves the best performance with the validation data. But in this work, our objective is to build a CNN model that fully explores the features extracted from a certain amount of training data and study how well the model generalizes to unseen (testing) data. Although we can obtain models with better performance by applying early stopping regularization, such models avoid overfitting issues by failing to learn the features from training data adequately. Consequently, we did not consider these cases and continued updating the CNN model parameters until the training loss converged. Furthermore, we do not consider transfer learning. Although transfer learning has the potential to significantly improve the sample efficiency of a learning agent, it obtains additional information from sources other than the training dataset. Hence, transfer learning has no direct connection with the sample complexity problem.

Table 4.1: CNN Architecture

Simple CNN					
layer	filter size	stride	channel	activation	batch norm
Conv	3×3	2	32	Leaky ReLU	Yes
Conv	3×3	2	32	Leaky ReLU	Yes
Conv	3×3	2	64	Leaky ReLU	Yes
FC	—	—	64	Leaky ReLU	No
FC	—	—	32	Leaky ReLU	No
Output	—	—	No. of classes	Softmax	No
Medium CNN					
layer	filter size	stride	channel	activation	batch norm
Conv	3×3	1	32	Leaky ReLU	Yes
Conv	3×3	2	32	Leaky ReLU	Yes
Conv	3×3	1	64	Leaky ReLU	Yes
Conv	3×3	2	64	Leaky ReLU	Yes
Conv	3×3	1	128	Leaky ReLU	Yes
Conv	3×3	2	128	Leaky ReLU	Yes
FC	—	—	256	Leaky ReLU	No
FC	—	—	64	Leaky ReLU	No
Output	—	—	No. of classes	Softmax	No
Complex CNN					
layer	filter size	stride	channel	activation	batch norm
Conv	3×3	1	32	Leaky ReLU	Yes
Conv	3×3	1	32	Leaky ReLU	Yes
Conv	3×3	1	64	Leaky ReLU	Yes
Conv	3×3	2	64	Leaky ReLU	Yes
Conv	3×3	1	128	Leaky ReLU	Yes
Conv	3×3	1	128	Leaky ReLU	Yes
Conv	3×3	2	128	Leaky ReLU	Yes
Conv	3×3	1	256	Leaky ReLU	Yes
Conv	3×3	1	256	Leaky ReLU	Yes
Conv	3×3	2	256	Leaky ReLU	Yes
FC	—	—	1024	Leaky ReLU	No
FC	—	—	256	Leaky ReLU	No
Output	—	—	No. of classes	Softmax	No

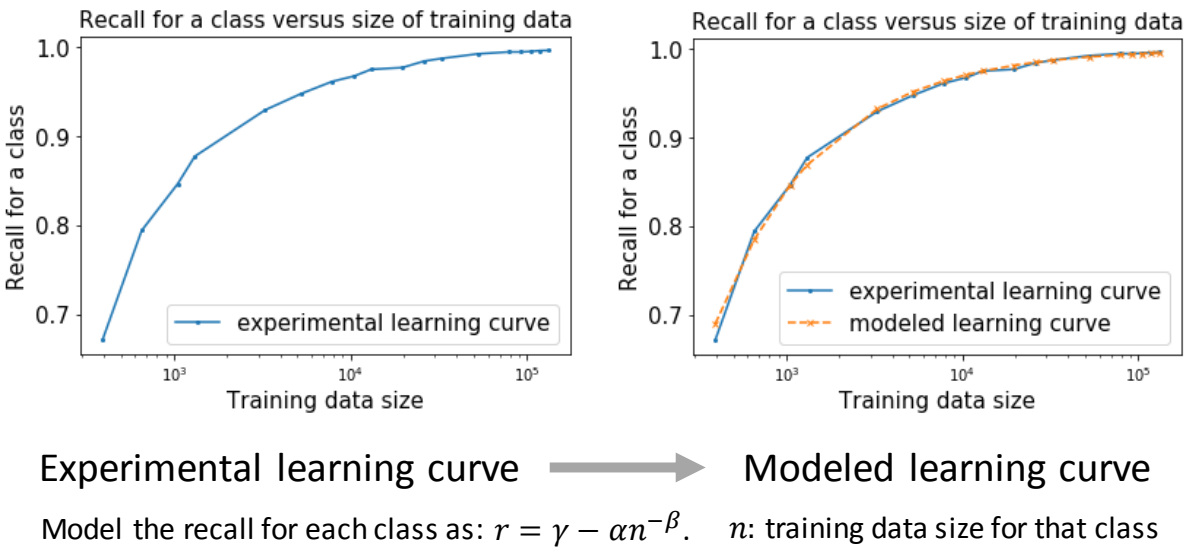


Figure 4.3: Experimental learning curve and modeled learning curve.

4.6 Learning Curve Generation and Modeling

We use *learning curves* to analyze the relationship between recall and the size of the training data. The x-axis of a learning curve represents the size of the training data for a particular class, and the y-axis represents the recall for that class. Note that our definition of learning curves is different from its conventional definition. In most of the prior works, a learning curve is defined as the number of training steps versus training/validation accuracy, while in some other works, it is defined as the size of training data versus testing accuracy.

4.6.1 Experimental Learning Curve

We first generated learning curves using experimental data, which we refer to as *experimental learning curves*, and then we used statistical models to fit those curves, which is referred to as *modeled learning curves*. Figure 4.3 illustrates examples of an experimental learning curve and a modeled learning curve.

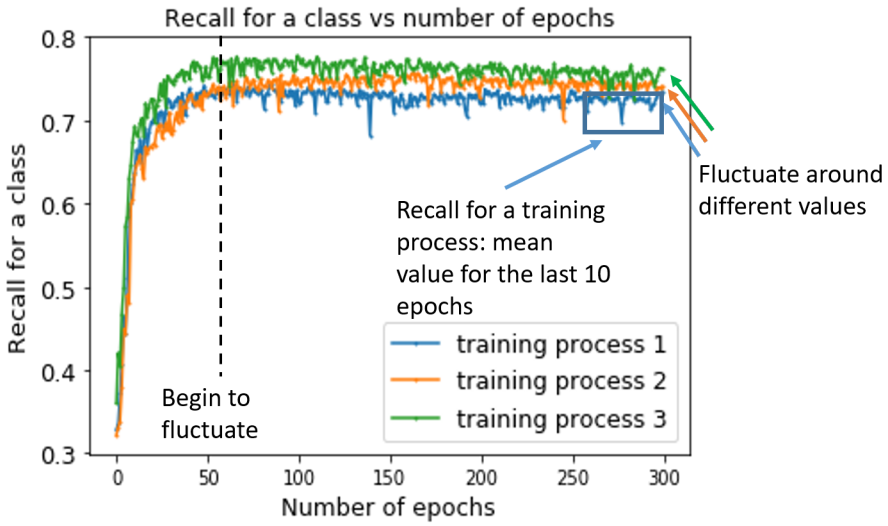


Figure 4.4: Recall for a class vs the number of epochs.

In both transmitter classification and modulation recognition, we randomly selected 20% of all the processed data points and used them as testing data. The remaining 80% of the data points were used as the training dataset. Each experimental learning curve is plotted with 20 (training data size, recall) pairs of points. To obtain the recall corresponding to a plotted point, we sampled a number of data points from the training dataset. Then we used them to train a CNN model and evaluate recall for all the classes using the testing dataset. The training dataset size in each experimental learning curve ranges from 0.3% to 100% of the overall training dataset size.

Since deep learning applies stochastic training methods to update parameters, there exists much randomness in each training process. Figure 4.4 illustrates how the recall for a class changes as the number of epochs increases. The recall fluctuates around certain values after some epochs. Moreover, the recall values for different training processes fluctuate around different values. Therefore, we computed the recall for each class in the following manner:

- For a given training process, we calculated the mean value of recall for the last 10 epochs after removing the maximum and minimum values.

- We computed the final recall by computing the average among recall values associated with 20 training processes after removing the maximum and minimum values. In each training process, the size of training data remains the same. But all the training data are resampled from the overall training dataset.

4.6.2 Modeled Learning Curve

To gain insights on how recall for each class varies with the size of training data, we need to come up with a model that can describe the experimental learning curves quantitatively. Hestness *et al.* [36] showed that in image and text processing applications, the testing error can be estimated using the inverse power-law model. Motivated by [36], we model the experimental learning curve using the *extended inverse-power law* model, which is the inverse power-law term plus a constant. For each class, the relationship between recall r and training data size n can be expressed as:

$$r = \gamma - \alpha n^{-\beta} + \epsilon, \tag{4.2}$$

where α, β are parameters that describe how fast recall increases with n and γ represents the upper bound of the recall. There is no irreducible classification error for a class if and only if $\gamma = 1$. We add a noise term ϵ to represent the discrepancy between the actual value and the estimated value. Due to the random sampling of training data and the stochastic training procedures, recall fluctuates around certain values for different training processes. The higher the variance of ϵ, the higher the uncertainty in the estimation of recall.

For each experimental learning curve, given the size of training data n_i and the corresponding recall $r_i, i = 1, 2, ..., N$, where N is the number of points in the experimental learning curve, our objective is to find α, β, γ to minimize the differences between the estimated recall values

and r_i. This optimization problem can be formulated as:

$$\min_{\alpha,\beta,\gamma} \sum_{i=1}^{N} |\alpha n_i^{-\beta} + \gamma - r_i|. \quad s.t. \quad \alpha > 0, \beta > 0, 0 < \gamma \leq 1.$$

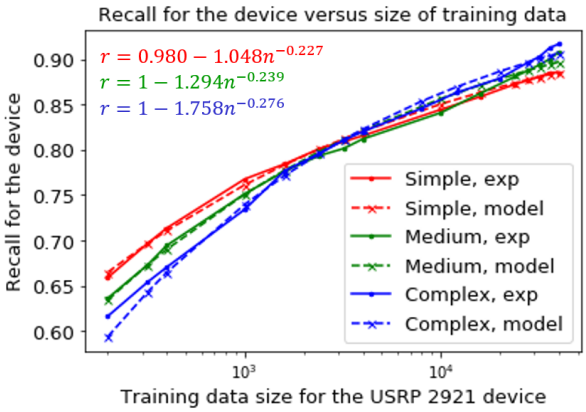
We minimize the summation of the absolute errors instead of the mean square errors. We chose this approach because absolute errors are more robust to outliers than mean square errors. Our aim is to ensure that the modeled learning curves can preserve the statistical features of the experimental learning curves even when outliers are present. We solved the optimization problem with sequential quadratic programming.

4.7 Experimental Results

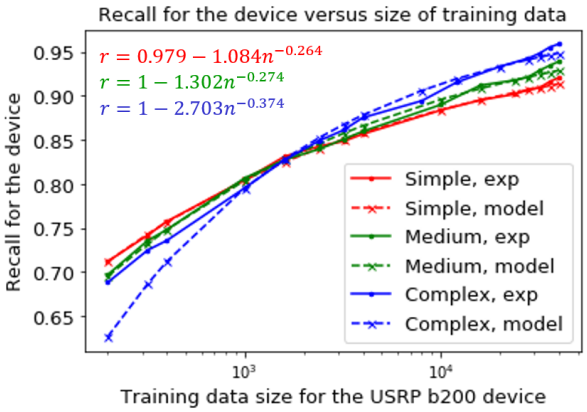
In this section, we provide a comprehensive analysis of how recall changes as the size of training data increases. First, we consider the case where the fraction of training data for each class remains the same as the size of training data increases. Next, we demonstrate how to estimate the training data size that is needed to achieve a certain performance target. In addition, we provide insights on balancing the trade-off between recall and training data size. Finally, we discuss how recall changes when the fraction of training data for each class changes as more training data is added.

4.7.1 Statistical Features of Learning Curves

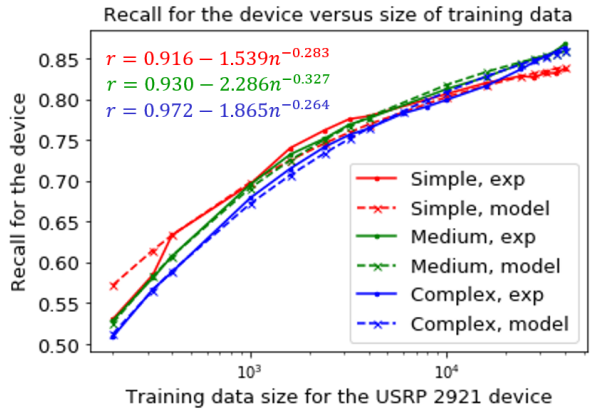
For transmitter classification, we present experimental learning curves and the corresponding modeled learning curves for one USRP 2921 device and one USRP b200 device under SRC and LOS scenarios in Figure 4.5. For modulation recognition, we present the learning curves



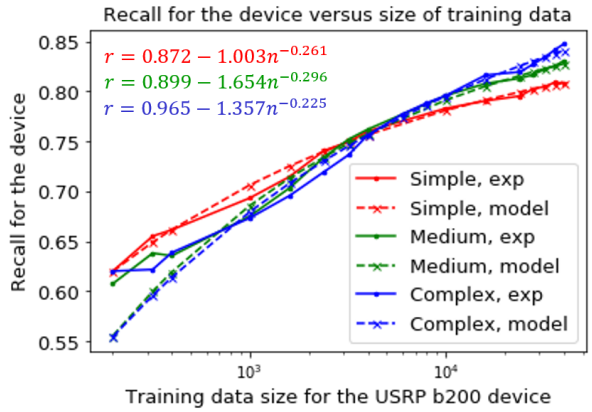
(a) USRP 2921, Channel: SRC.



(b) USRP b200, Channel: SRC.



(c) USRP 2921, Channel: LOS.

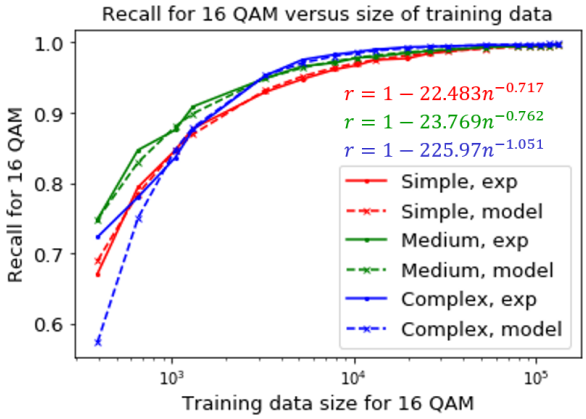


(d) USRP b200, Channel: LOS.

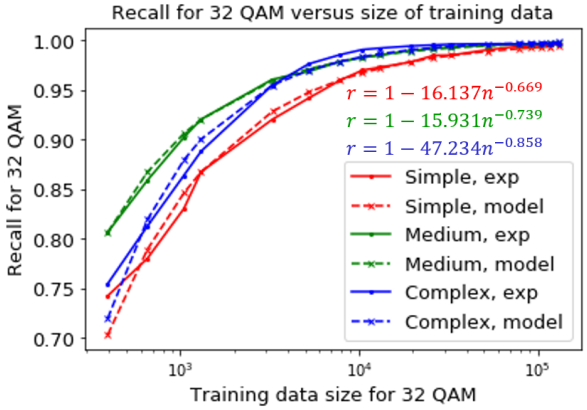
Figure 4.5: Experimental and modeled learning curves for transmitter classification using different CNN models.

associated with 16 QAM and 32 QAM when SNR is 12–20 dB and 2–10 dB in Figure 4.6. The legend for each figure indicates the CNN model and the type of learning curve—i.e., experimental learning curve (exp) or a modeled learning curve (model). The equations that define the modeled learning curves are given in each figure as well. The x-coordinates are plotted in log-scale for better illustration.

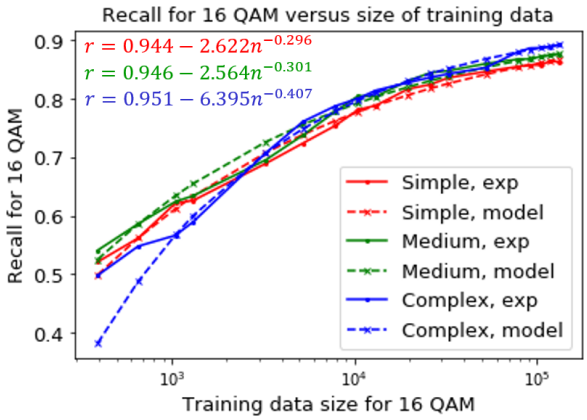
Observations from experimental learning curves. By comparing the experimental learning curves in Figure 4.5 and Figure 4.6, we make the following observations. When



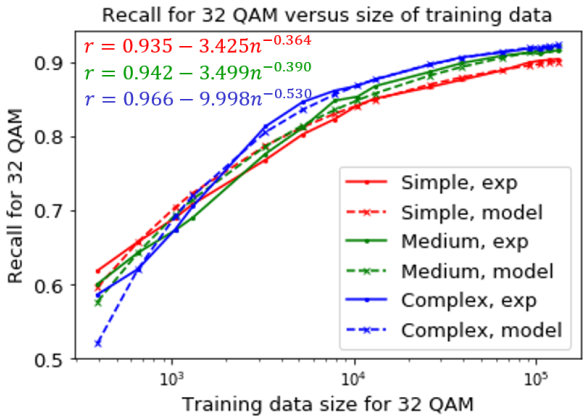
(a) 16 QAM. SNR: 12-20 dB.



(b) 32 QAM. SNR: 12-20 dB.



(c) 16 QAM. SNR: 2-10 dB.



(d) 32 QAM. SNR: 2-10 dB.

Figure 4.6: Experimental and modeled learning curves for modulation recognition using different CNN models.

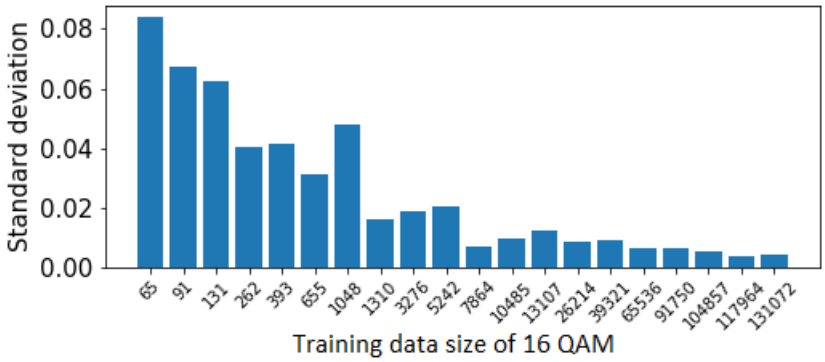


Figure 4.7: Standard deviation of recall versus the size of training data.

we compare the experimental learning curves of two models of different complexities for the same class, there exists a threshold n_T, such that recall for the more complex model is always higher than the simpler model when trained with more than n_T data. For example, in Figure 4.5a, in the first half of the experimental learning curves, the Simple CNN achieves a higher recall than the other two CNN models. However, when there are more than 20k training data, the Complex CNN outperforms the Medium CNN, and the Medium CNN outperforms the Simple CNN. Such a phenomenon occurred because deep learning models with a higher complexity can extract more features given a sufficient amount of training data, which is beneficial in increasing the recall. But without enough training data, some of the extracted features may not generalize well to unseen testing data in high complexity models due to overfitting. In some cases, a high complexity model will always achieve a higher recall than a low complexity model, even when such high complexity models are trained with a small amount of training data, as illustrated in Figure 4.6b (where we compared the Medium/Complex CNN with the Simple CNN). This is a special case where n_T can be set to an arbitrary value while the conclusion still remains valid.

Intrinsic features of modeled learning curves. To quantitatively analyze the relationship between recall and the training data size, we modeled all experimental learning curves using the extended inverse power-law models. Note that γ represents the upper bound of the recall, as explained in section 4.6.2. The exponential parameter β dominates the rate at which the recall is increased with the training data size. By analyzing the modeled learning curve parameters presented in Figure 4.5 and Figure 4.6, we make the following observations:

1. The recall may not be estimated correctly using the extended inverse-power law model when the size of training data is small. In our experiments, the recall for an experimental learning curve is calculated by taking the average values across 20 training processes. When the variance is too high, the estimated average may deviate significantly from the true

average. In Figure 4.7, we demonstrate how the standard deviation of the recall changes as the size of training data increases. Data was recorded based on recall for 16 QAM when SNR is between 12 dB and 20 dB. Standard deviation is calculated by training the Medium CNN 20 times. A general trend is that the standard deviation decreases as the training dataset size increases. When the training dataset size is not large enough, the standard deviation can be higher than 0.05, which indicates that the actual recall is very likely to differ from the estimated value by 0.1 (two standard deviations). Note that the recall ranges from zero to one, a difference of 0.1 makes the estimation of recall unreliable and not useful. In all of our experiments, the modeled learning curves fit well with the experimental learning curves when the size of the training data for each class exceeds 1000.

2. We use absolute error between the exact recall value obtained from the experimental learning curve and the estimated recall value from the modeled learning curve to evaluate the goodness of fit. For each pair of experimental and modeled learning curves, the average and maximum errors are calculated when the size of the training data for each class exceeds 1000. From Figure 4.5 and Figure 4.6, we observe that the modeled learning curves fit very well with the experimental learning curves. The average errors range from 0.192% to 0.762% in all the experiments. The maximum errors range from 0.281% to 1.867% in all the experiments. Such results indicate that the recall for each class can be estimated precisely using the extended inverse-power law model.

3. By comparing the modeled learning curves for the same class, we find that an increase in the complexity of CNN increases the value of γ. When a sufficient amount of training data is available, a CNN model with higher complexity has a higher model capacity and is expected to achieve better performance due to the reduction of irreducible classification errors. In other words, increasing model complexity is beneficial in decreasing irreducible classification errors.

4. When there is no irreducible classification error, the model with higher complexity results in a higher value of β. Such a phenomenon can be observed by comparing Figure 4.5a, 4.5b, 4.6a and 4.6b. Our understanding of this phenomenon is that a CNN model with higher complexity updates more model parameters to extract features when the training data size increases, which makes the recall increase at a faster pace. Nevertheless, this observation cannot be inferred for more general scenarios when there exist irreducible classification errors. For example, in Figure 4.5c and Figure 4.5d, the Complex CNN has a smaller value of β when compared with the other two CNN models. A decrease in the irreducible classification error could result in a decrease of β. In other words, the value of β is impacted by both factors in a complicated manner.

Relationship between recall for each class and testing accuracy. Prior works [36, 70] studied the sample complexity problem for special cases in which the testing dataset is fixed. They concluded that the testing accuracy follows the inverse-power law model with the training data size. In contrast, we have proposed model-intrinsic metrics and studied the sample complexity problem for general cases in which the testing dataset is not necessarily fixed. In this part, we demonstrate that our findings imply conclusions from prior works. In other words, if a model is evaluated with a fixed testing dataset, then recall and training dataset size for each class follow the extended inverse-power law model implies that accuracy and training dataset size follow the extended inverse-power law model.

Denote n as the size of the overall training dataset, n_i as the training data size for class i, and r_i as the recall for class i. We assume that the ratio n_i/n is a constant for all $i \in \mathcal{Y}$ as n increases. Then r_i and n_i follows the extended inverse-power law model implies that r_i and n follows the extended inverse-power law model. Hence, r_i can be expressed as $r_i = \gamma_i - \alpha_i n^{-\beta_i}$. Suppose there are n_i^{test} data points that belong to class i in the testing dataset, then the

testing accuracy is:

$$\begin{aligned}\text{Acc} &= \frac{\sum_{i=1}^{M} n_i^{\text{test}} r_i}{\sum_{i=1}^{M} n_i^{\text{test}}} = \frac{\sum_{i=1}^{M} n_i^{\text{test}} (\gamma_i - \alpha_i n^{-\beta_i})}{\sum_{i=1}^{M} n_i^{\text{test}}} \\ &= \frac{\sum_{i=1}^{M} n_i^{\text{test}} \gamma_i}{\sum_{i=1}^{M} n_i^{\text{test}}} - \frac{n^{-\beta_k} (\sum_{i \neq k} n_i^{\text{test}} \alpha_i n^{\beta_k - \beta_i} + n_k^{\text{test}} \alpha_k)}{\sum_{i=1}^{M} n_i^{\text{test}}}.\end{aligned}$$

where $k = \arg\min \beta_i, i \in \mathcal{Y}$. Since $\beta_k < \beta_i$ when $i \neq k$, $n^{\beta_k - \beta_i}$ is significantly lesser than 1 for sufficiently large n. Therefore, the testing accuracy can be approximated as:

$$\text{Acc} \approx \frac{\sum_{i=1}^{M} n_i^{\text{test}} \gamma_i}{\sum_{i=1}^{M} n_i^{\text{test}}} - \frac{n_k^{\text{test}} \alpha_k}{\sum_{i=1}^{M} n_i^{\text{test}}} \cdot n^{-\beta_k}. \tag{4.3}$$

When testing dataset is fixed, n_i^{test} are constants for $\forall i \in \mathcal{Y}$ and eq. (4.3) indicates that the testing accuracy and n follow the inverse-power law model. This justifies our assertion that prior work conclusions can be implied from this work.

4.7.2 Application of Learning Curves

We can estimate how much training data is needed to achieve a target performance with the help of the modeled learning curve. For example, let us assume that we have n' training data points for a class. We have a CNN model trained with those data points, but the recall for this class is lower than the required value of r. We first generate an experimental learning curve by training the model using subsets of training data. Then we use the extended inverse power-law model to generate the modeled learning curve. The amount of required training data for that class can be estimated as $n \approx (\frac{\gamma - r}{\alpha})^{-\frac{1}{\beta}}$ if the proportion of the training data size for each class remains the same when more training data is added. Researchers can also take the variance into consideration. A smaller amount of training data is needed when the best CNN model is selected after training the model multiple times. On the other hand, a

greater amount of training data is needed if the recall needs to be higher than r with a very high probability.

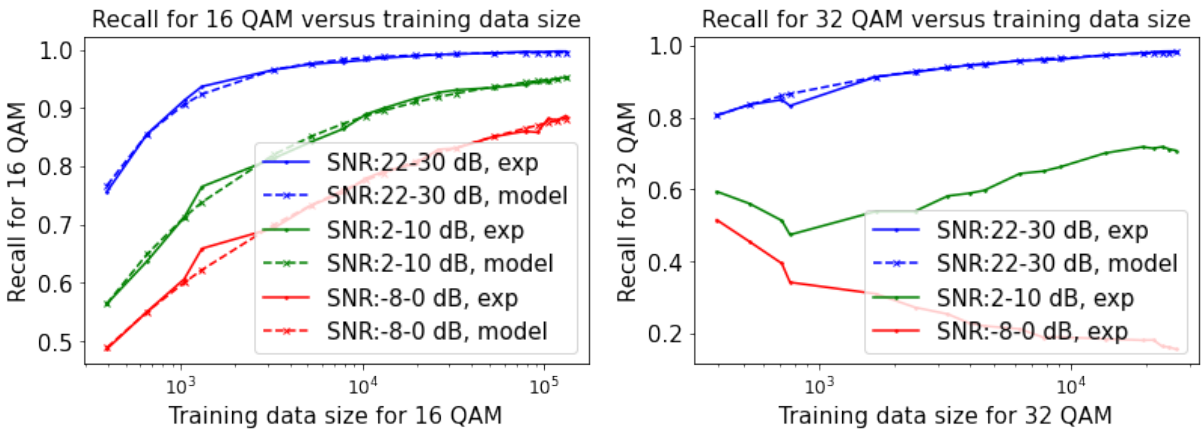
We can also use the modeled learning curve to balance the trade-off between recall and the size of the training data. We observe that when the training data size is small, adding little training data can significantly improve the CNN performance. However, the performance gain drops as more training data is added. After a certain point, the rate at which recall increases slows down significantly.

The slope of a modeled learning curve is defined as:

$$m = \frac{\Delta\text{recall}}{\Delta n} \approx \alpha\beta \cdot n^{-(\beta+1)}. \tag{4.4}$$

which represents how fast the recall increases when more training data is added. We use the first-order derivative from the modeled learning curves to approximate the slope. The slope is a decreasing function, which conforms to the fact that adding more training data becomes less effective. For instance, the slope $m = 10^{-6}$ at $n = n_0$ indicates that recall cannot be increased by 1% with less than 10K training data when the training data size exceeds n_0.

In practice, the amount of training data that can be acquired is constrained by time and resources. Hence, having a better understanding of the relationship between recall and the training dataset size is an important step towards the effective application of deep learning to most problems. The slope of the modeled learning curve can be used to decide when to stop adding more training data due to the limited performance gain. First, users set a threshold of the slope based on the upper bound of the additional amount of training data they are willing to generate (Δn) in order to increase recall by a certain level (Δrecall). Then estimate the training data size n_0 based on eq. (4.4). Users can stop generating new training data when the training data size exceeds n_0.



(a) 16 QAM. (b) 32 QAM

Figure 4.8: How recall changes when the proportion of the training data size for each class changes as more training data is added.

4.7.3 Changing the Fraction of Training Data for Each Class

We conducted the following binary classification experiment to illustrate how recall changes if the fraction of training data for each class does not remain the same when more training data is added. The task is to classify 16 QAM and 32 QAM using the Medium CNN in the following three scenarios:

- SNR ranges from 22 dB to 30 dB. There are almost no irreducible classification errors.
- SNR ranges from 2 dB to 10 dB. There exist a small portion of irreducible classification errors.
- SNR ranges from -8 dB to 0 dB. There exist a large portion of irreducible classification errors.

Similar to the previous setup, we randomly selected 20% of all the processed data points and used them as testing data. Training data is sampled from the remaining 80% of the

data points. Initially, the sizes of the training data for both classes are the same. Then we increase the training data size for 16 QAM at a higher rate than 32 QAM, so that the proportion of 16 QAM training data also increases. We increase the size of training data for 16 QAM from 0.4k to 13k, while we increase the size of the training data for 32 QAM from 0.4k to 2.6k. The recall values are calculated using the same approach as section 4.6. To alleviate the data imbalance issue during the training process (i.e., the update of the model parameters is dominated by the majority class), we maintained the same number of training data points for both classes in each batch. We did not take other data upsampling and downsampling techniques into consideration. We leave it as part of our future work. The learning curves for both modulation schemes at different SNR values are illustrated in Figure 4.8. It is obvious that when SNR ranges from 2 to 10 dB and when SNR ranges from -8 to 0 dB, the experimental learning curves for 32 QAM do not follow the extended inverse-power law model. Hence, we do not present the corresponding modeled learning curves. From Figure 4.8, we can make the following observations:

1. Although the proportion of training data for each class keeps changing as more training data is added, the extended inverse-power law model still applies when SNR ranges from 22 dB to 30 dB. We observe the same trend in multiple experiments that use all the three CNN models and increase the training data size for each class in different ways. In addition, this conclusion generalizes transmitter classification under the SRC channel condition when we use the Medium and Complex CNN. Note that in section 4.3, we conjecture that if no irreducible classification errors occur using the given deep learning model, the extended inverse-power law model applies even though the fraction of training data for each class keeps changing. Our experimental results conform to this conjecture. Since there are infinitely many ways to increase the training data size for each class, we could not justify this conjecture in all the possible scenarios. But we expect it to be valid in other untested scenarios.

2. When there exist irreducible classification errors, the inverse-power law model is not applicable if the fraction of training data for each class does not remain the same as more training data is added. Compare the three experimental learning curves in Figure 4.8b, we can observe that a larger portion of irreducible classification errors makes the experimental curve deviate more from the extended inverse-power law. When SNR ranges from -8 dB to 0 dB, the recall continues to decrease as more unbalanced training data is added. Such results conform to the analysis shown in section 4.3 as well.

In this work, we focus more on generalized cases where both reducible and irreducible classification errors occur because: (1) Researchers cannot keep increasing the model complexity to reduce the irreducible classification errors because it may cause severe overfitting issues given a limited amount of training data. (2) In real-world wireless communications applications, there always exist irreducible classification errors due to the dynamic features of the wireless channel conditions. As a consequence, we assert that a sufficient condition to estimate recall reliably in general cases using the extended inverse-power law model is that the fraction of training data for each class remains the same when more training data is added to improve model performance based on previous discussions.

4.8 Chapter Summary

Understanding the relationship between model performance and the training data size is more important in wireless communications than other domains since wireless communication tasks are more propagation environment-specific. In this chapter, we analyze the relationship between recall for each class and training data size quantitatively. The learning curves can be used to estimate the required training data size for a performance target. Moreover, we summarize some sufficient conditions for which recall can be estimated accurately using the

extended inverse-power law model.

Nevertheless, our finding is only the tip of the iceberg. We are only able to describe the relationship between model performance and training sample size quantitatively under several constraints, such as no irreducible errors exist or the fraction of training sample size for each class remain constants. It remains unclear how recall changes with training data size in more general scenarios. In addition, most of the results in this work are based on empirical results and lack theoretical supports.

In the future, we will analyze how the learning curves change if upsampling and downsampling techniques are applied to handle the data imbalance issue. Moreover, it could be very promising if we can find theoretical explanations why the recall for each class and the training data size for each class follow the inverse-power law model. Finally, it will be interesting to analyze how the model performance changes when the size of training data for each class is added arbitrarily.

Chapter 5

Detecting Out-of-Distribution Data

5.1 Introduction

When deploying deep learning products in real-world applications, there is very little or no control over the testing data. Some testing data can be anomalous or significantly different from the data used in training. For example, one challenging application of deep learning is bacteria identification, since bacteria keep evolving and will inevitably contain genomes from unseen classes not present in the training data [86]. These testing data cause the deep learning model to generate highly unreliable results.

Prior works have shown that deep learning models tend to make high confidence predictions even for completely unrecognizable [68] or irrelevant inputs [64]. The presence of OOD data can negatively impact the analysis of results in real-world applications. In safety-critical and security-critical applications such as autonomous driving, OOD data may result in unexpected perception errors and cause accidents or system failures. Since OOD data do not result in explicit errors in the model, such an impact may go undetected. Hence, the successful deployment of deep learning-based systems requires that the system be able to distinguish between OOD and ID data.

As illustrated in Figure 5.1, we categorize OOD data into the following two types:

- Type 1: Data that do not belong to any of the existing classes. This type of OOD

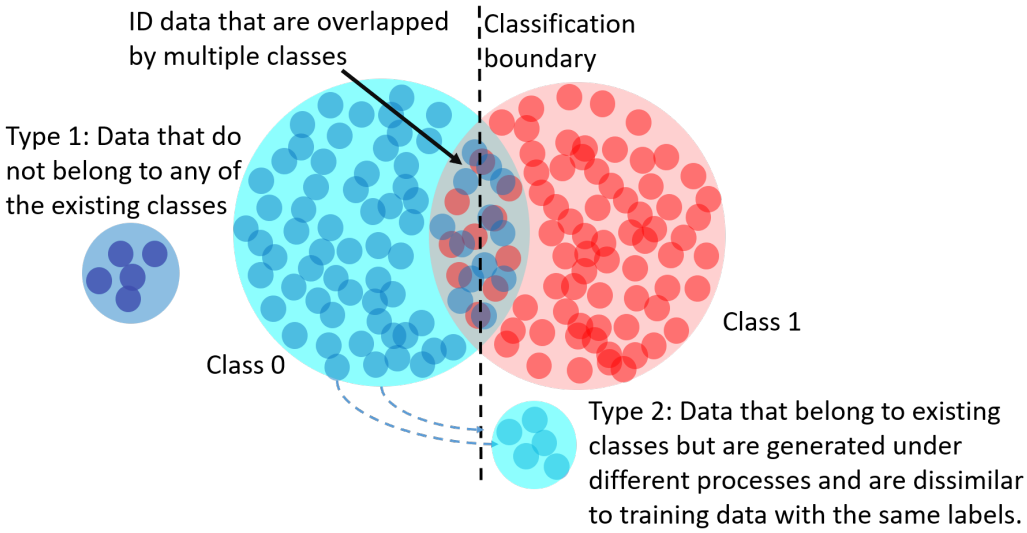


Figure 5.1: Two types of OOD data in classification algorithms.

data causes incorrect classification decisions because a classifier erroneously classifies such irrelevant data to an existing class.

- Type 2: Data that belong to the same classes in the training data, but are generated under different processes and are dissimilar to training data with the same labels. For example, the testing data are blurred cat images whereas all training data are high fidelity animal images. Such OOD data are highly likely to be classified incorrectly because of the uncertainty of applying the features extracted from ID data to them.

In image and text domains, researchers are more interested in detecting Type 1 OOD data. Type 2 OOD data is uncommon and has received little to no attention. However, Type 2 OOD data is very pervasive in wireless communications applications due to the diverse channel conditions experienced during data transmission. Even though researchers strive to collect the training data as comprehensively as possible, the radio propagation data can still be transmitted under a new channel condition. In real-world wireless communications applications, RF signals can be easily interfered with by unexpected signals and noise. In

channel-sensitive applications such as transmitter classification, a small change in the wireless channel can increase the classification errors dramatically.

When the dimension of input data is high, it is infeasible to estimate the distribution of ID data analytically. Prior works demonstrated two methods to detect OOD data in classification algorithms: (1) set a threshold for confidence score, which is defined as the maximum value in the softmax layer [34, 53], (2) analyze the penultimate layer of a DNN [50]. However, both approaches have limitations in detecting OOD data in wireless communications applications. The first method always incorrectly detects ID data overlapped by multiple classes (the middle region in Figure 5.1) as OOD. Moreover, we frequently observed that certain forms of OOD data have higher confidence scores than ID data. The second approach works if the data from the penultimate layer follows the multivariate Gaussian distribution. Such an assumption is not always valid when we process radio propagation data.

Recent works have applied VAE [42] to perform anomaly detection [1, 106] and OOD data detection [18, 110]. Nevertheless, VAE is an unsupervised generative model and cannot be used for classification. If we choose to use VAE to detect OOD data in classification, we need to build two separate models. One is the VAE-based deep learning model to detect OOD data, and the other is a standard CNN or DNN model to perform classifications. Hence, such approaches increase the computation overhead and resource usage significantly.

To overcome these issues, we propose a new deep learning-based model called FOOD (**F**eature representation for detecting **OOD** data). The architecture of FOOD borrows the idea of VAE. FOOD first maps high-dimensional input data to low-dimensional feature representations in the latent space. The feature representations contain essential information that can be used to reconstruct the input data and the information related to input data labels. Then FOOD classifies the input data based on the feature representations and determines whether it is OOD by analyzing both the feature representations and the data reconstruc-

tion errors. Compared with prior VAE-based OOD data detection models, FOOD has the following advantages: (1) FOOD incorporates class labels of the training data in the training process. Therefore, FOOD is a unified model that is able to perform the classification and OOD data detection simultaneously. (2) FOOD embeds label information in the feature representations and combines two metrics to detect OOD data. Such an approach makes the feature representations of OOD data more distinguishable from ID data and outperforms other VAE-based models that only use reconstruction probabilities [1] to detect OOD data.

We evaluate the performance of FOOD on transmitter classification experiments and modulation recognition experiments. In spectrum access systems, malicious secondary users that violate the spectrum sharing rules or cause interference with the primary users can be identified through transmitter classification. However, if the testing data are OOD, the system may misclassify benign transmitters as malicious. Modulation recognition deals primarily with identifying modulation schemes based on the constellation points. Failure to detect OOD data may result in unpredictable system failures.

Our contributions are summarized as follows:

- We propose a novel deep learning model, named FOOD, to detect OOD data in wireless communications applications. FOOD analyzes both low-dimensional feature representations and reconstruction errors of the input data, which significantly increases the OOD data detection accuracy when compared with previous works.
- We evaluate the performance of FOOD using two datasets that apply completely different signal processing algorithms. It demonstrates the potential of FOOD to be applied to various wireless communications applications.
- To the best of our knowledge, we are the first to systematically study the impact and detection of OOD data in wireless communications applications.

5.2 Related Work

Besides using the confidence score or the penultimate layer, there have been other efforts toward developing efficient OOD data detection methods in classification. Several prior works propose to add some OOD data in the training process. For example, Hendrycks *et al.* [35] trained their model using both ID training data and data sampled from different datasets. Lee *et al.* [49] added samples generated by a generative adversarial network (GAN) trained with the same training dataset to strengthen the OOD detection capability. However, the robustness of such methods is in question since there is no guarantee that similar results can be generalized to OOD data from other sources. Also, adding OOD data from other datasets during training is not tenable since this is not usually available to the model developer at the training stage. Ensemble methods using multiple models can also be used to detect OOD data [48, 99], but are usually very computation-intensive.

In unsupervised learning, OOD data detection can be performed by evaluating the reconstruction probability [1] or log-likelihood [97] of inputs under a generative model. Prior works have demonstrated the practicability of applying VAE to detect OOD data based on reconstruction probabilities in image processing [1, 111] and web applications [106]. However, Nalisnick *et al.* [67] noticed that generative networks such as VAE and GLOW [43] that were trained on the CIFAR-10 dataset assigned higher likelihood to the SVHN dataset. This result is worrisome since CIFAR-10 and SVHN contain significantly different images. Such phenomena significantly degrade the ability to detect OOD data based on log-likelihood. Several countermeasures [15, 86] have been proposed to alleviate this issue.

OOD data detection in wireless communications applications was mentioned in [60] using dimension reduction methods. However, such an approach uses simplified statistical assumptions and the performance was only evaluated on classifying distinguishable Wi-Fi devices.

Besides, the impact of channel variations was not studied. The detection of unknown RF signal types was discussed in [91] using k-means and the minimum covariance determinant method. But both approaches are effective only under the condition that the known RF signal data and unknown RF signal data form spherical clusters or close to multi-variate Gaussian distributions, which may not be true in general scenarios.

5.3 Overview of FOOD

In this section, we provide an overview of FOOD and briefly explain how FOOD performs classification and detects OOD data. The mathematical and implementation details of FOOD are explained in-depth in sections 5.4 and 5.5, respectively. In the rest of this paper, we use the following notations:

- $\mathbf{x}_{\text{in}}$: input data. Depending on the context, they can be either referred to as ID data or OOD data.
- $y \in \mathcal{Y} = \{1, 2, \cdots, M\}$: the label y for input data $\mathbf{x}_{\text{in}}$. $\mathcal{Y}$ is the set of labels and M is the number of classes.
- $\mathbf{z}$: a random variable in the latent space. It can be regarded as a feature representation of the input data.
- z_{dim}: the dimension of $\mathbf{z}$.

The architecture of FOOD is demonstrated in Figure 5.2. An illustration of FOOD procedures is shown in Figure 5.3. Similar to VAE, FOOD is also composed of an encoder and a decoder. The encoder maps the high-dimensional input data $\mathbf{x}_{\text{in}}$ to two vectors $\mu(\mathbf{x}_{\text{in}})$ and $\Sigma(\mathbf{x}_{\text{in}})$. These two vectors determine $q(\mathbf{z}|\mathbf{x}_{\text{in}}) = \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$, which is the probability

distribution of the low-dimensional feature representations $\mathbf{z}$ given $\mathbf{x}_{\text{in}}$. $\boldsymbol{\mu}_x$ is equal to vector $\mu(\mathbf{x}_{\text{in}})$, and $\boldsymbol{\Sigma}_x$ is a diagonal matrix whose elements equal to $\Sigma(\mathbf{x}_{\text{in}})$. The decoder is able to reconstruct input data $\mathbf{x}_{\text{in}}$ from samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ with high fidelity.

A key property that differentiates FOOD from a standard VAE is that the distributions $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ can be used to perform classification tasks and detect OOD data. A standard VAE does not utilize labels of the data and $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ for all inputs $\mathbf{x}_{\text{in}}$ are designed to be close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Therefore, the samples of $\mathbf{z}$ from all classes are clustered together, which cannot be used for classification. More importantly, we find that VAE maps almost all inputs, including random noise, to distributions very close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Such a phenomenon makes the samples $\mathbf{z}$ useless in detecting OOD data. FOOD overcomes these issues using following steps. First, FOOD tries to make $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ close to a predefined distribution $p(\mathbf{z}|\mathbf{x}_{\text{in}}) = \sum_{k=1}^{M} p(\mathbf{z}|k) \cdot \mathbf{1}_{y=k}$ in the training stage, where y is the label of $\mathbf{x}_{\text{in}}$. Since each $\mathbf{x}_{\text{in}}$ in the training dataset has a unique label, $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ takes on exactly one of the distributions $p(\mathbf{z}|k)$, $k \in \mathcal{Y}$ based on the label of $\mathbf{x}_{\text{in}}$. Distributions $p(\mathbf{z}|k)$ for different k are designed to have very little overlap with each other. Therefore, for $\mathbf{x}_{\text{in}}$ with the same labels, distributions $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ are "close" to each other. While for $\mathbf{x}_{\text{in}}$ with different labels, $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ are "very different" from each other. Such a property is illustrated in Figure 5.3. As a result, samples $\mathbf{z}$ from data with the same label are clustered together, and clusters associated with different labels are separate from each other. This enables FOOD to classify $\mathbf{x}_{\text{in}}$ by analyzing $q(\mathbf{z}|\mathbf{x}_{\text{in}})$.

FOOD classifies $\mathbf{x}_{\text{in}}$ as ID data if and only if the following two criteria are satisfied: (1) Samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ are close to one of the clusters associated with the training labels. (2) The decoder can reconstruct $\mathbf{x}_{\text{in}}$ from the samples of $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ with small errors. We assert that FOOD is able to successfully detect most forms of OOD data by applying the aforementioned two criteria. First, most forms of OOD data generate samples $\mathbf{z}$ that are located far from the clusters formed by ID data. Such OOD data can be distinguished by

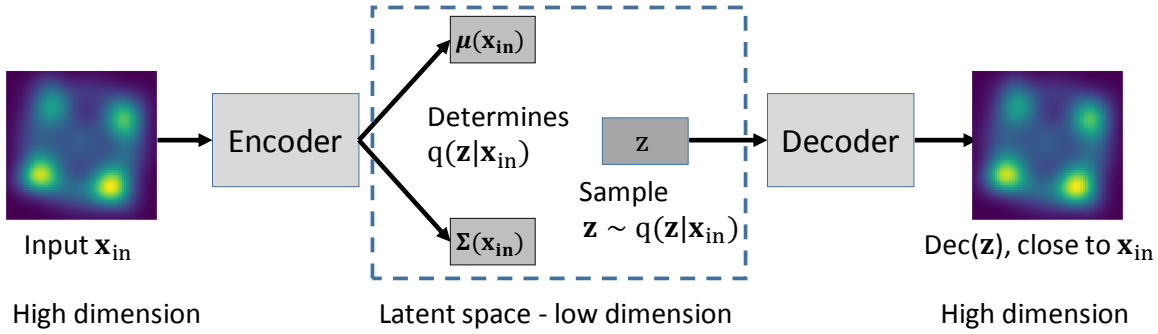


Figure 5.2: Architecture of FOOD.

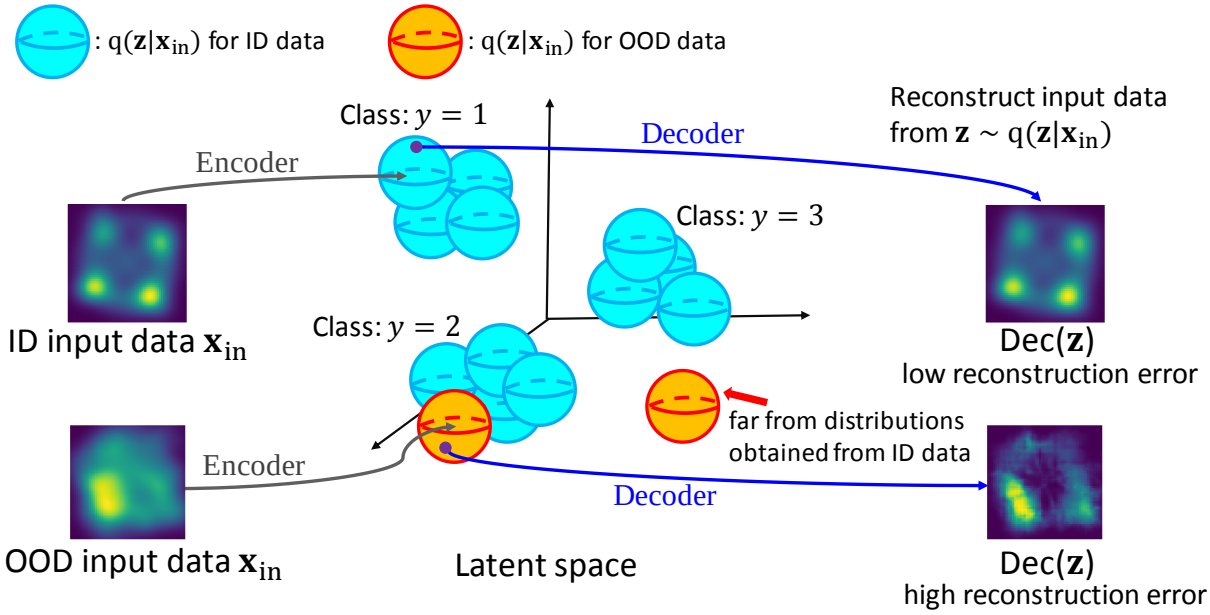


Figure 5.3: Illustration of FOOD procedure.

analyzing the statistical distances between $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ and those clusters. Second, since the encoder is a many-to-one mapping, some OOD data may generate distributions $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ close to those from ID data. However, the decoder is trained to generate outputs similar to ID data from samples drawn from such distributions, which results in high reconstruction errors for these OOD data, as demonstrated in Figure 5.3. On the other hand, FOOD does not incorrectly detect ID data overlapped by multiple classes as OOD data. This is because

such ID data only result in overlap among clusters formed by samples $\mathbf{z}$ from different classes. Such phenomena do not cause errors based on our OOD data detection criteria.

In the machine learning community, data sampled from a *sufficiently* different distribution from training data are denoted as OOD data. *FOOD, as well as other OOD data detectors, are not designed to detect the data generated by different processes but have strong similarities with ID data.* For instance, we will demonstrate in section 5.7 that data points corresponding to 16 PSK and 32 PSK are very hard to distinguish in modulation recognition. If we take one modulation scheme as ID and the other one as OOD, FOOD cannot detect such OOD data accurately. Besides, a change of radio propagation channel does not necessarily result in dissimilar signal data received by a receiver. In the DeepSig dataset for modulation recognition, most of the constellation diagrams (and density plots) for SNR=12 dB are similar to those for SNR=30 dB. Although such data are generated under different processes from the training data, the classification accuracy is similar to the ID testing data. Failing to detect such data as OOD should not be considered as failures of OOD detectors.

5.4 In-Depth Analysis of FOOD

5.4.1 Loss Function of FOOD

As illustrated in Figure 5.2 and Figure 5.3, the function of the encoder is to map $\mathbf{x}_{\text{in}}$ to distribution $q(\mathbf{z}|\mathbf{x}_{\text{in}})$, and the function of the decoder is to generate $\text{Dec}(\mathbf{z})$ that is close to $\mathbf{x}_{\text{in}}$ given $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$. In practice, it is almost impossible for $\text{Dec}(\mathbf{z})$ to be precisely equal to $\mathbf{x}_{\text{in}}$. As a consequence, we further model the output data $\mathbf{x}_{\text{out}}$ as $\mathbf{x}_{\text{out}} = \text{Dec}(\mathbf{z}) + \boldsymbol{\eta}$, where $\boldsymbol{\eta}$ denotes perturbations added to $\text{Dec}(\mathbf{z})$. The most common choice of $\boldsymbol{\eta}$ is Gaussian distribution and the output distribution is expressed as $p(\mathbf{x}_{\text{out}}|\mathbf{z}) = \mathcal{N}(\text{Dec}(\mathbf{z}), \sigma^2\mathbf{I})$, in which σ is

a hyperparameter. Mathematically, making $\text{Dec}(\mathbf{z})$ close to $\mathbf{x}_{\text{in}}$ is equivalent to maximizing the probability $p(\mathbf{x}_{\text{out}} = \mathbf{x}_{\text{in}}|\mathbf{x}_{\text{in}})$. For convenience sake, we use $\mathbf{x}_{\text{out}}$ to refer to $\mathbf{x}_{\text{out}} = \mathbf{x}_{\text{in}}$ in all the equations in the rest of this paper.

After applying Bayes theorem to $p(\mathbf{z}|\mathbf{x}_{\text{out}}, \mathbf{x}_{\text{in}})$ and some mathematical transformations, we can derive that $p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$, $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ and $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ satisfy the following inequality when $\mathbf{z}$ follows the distribution $q(\mathbf{z}|\mathbf{x}_{\text{in}})$, which we refer to as the Evidence Lower Bound (ELBO):

$$\log p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}}) \geq \mathbb{E}_{\mathbf{z}} \left[\log p(\mathbf{x}_{\text{out}}|\mathbf{z})\right] - \text{KL}\left[q(\mathbf{z}|\mathbf{x}_{\text{in}})||p(\mathbf{z}|\mathbf{x}_{\text{in}})\right]$$

where KL denotes the KL-divergence.

Since it is intractable to maximize $\log p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$ directly, we can maximize its lower bound instead. The first term of ELBO represents the expectation of the log-likelihood $\log p(\mathbf{x}_{\text{out}}|\mathbf{z})$. Recall that $\mathbf{x}_{\text{out}} = \text{Dec}(\mathbf{z}) + \boldsymbol{\eta}$, we have:

$$p(\mathbf{x}_{\text{out}}|\mathbf{z}) \propto \exp\left(-\frac{\|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z})\|^2}{2\sigma}\right).$$

which indicates that maximizing $p(\mathbf{x}_{\text{out}}|\mathbf{z})$ is equivalent to minimizing the mean square error $\|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z})\|^2$. The mean square error is also referred to as the reconstruction error.

The second term in ELBO represents the KL-divergence between $q(\mathbf{z}|\mathbf{x}_{\text{in}})$, the distribution of $\mathbf{z}$ learned from the encoder, and our pre-defined distribution $p(\mathbf{z}|\mathbf{x}_{\text{in}})$. In the training process, for $\mathbf{x}_{\text{in}}$ with class label $k, k \in \mathcal{Y}$, we choose $p(\mathbf{z}|\mathbf{x}_{\text{in}}) = p(\mathbf{z}|k) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. Covariance matrices $\boldsymbol{\Sigma}_k$ are all set to be identity matrix $\mathbf{I}$. Based on our observations, it is preferable to set vectors $\boldsymbol{\mu}_k$ such that the Euclidean distances $\|\boldsymbol{\mu}_k - \boldsymbol{\mu}_i\|_2$ are the same for any $k \neq i$. In FOOD, the $2k-1$ and the $2k$ coordinates of $\boldsymbol{\mu}_k$ are equal to 5, and the other coordinates are all set to be zeros (e.g., $\boldsymbol{\mu}_2 = [0, 0, 5, 5, 0, \cdots, 0]$). In section 5.5, we will provide guidelines

on parameter settings and give more explanations on why we chose $\boldsymbol{\mu}_k$ as described above.

Since both $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ and $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ follow multivariate Gaussian distributions, for $\mathbf{x}_{\text{in}}$ belongs to class k, the KL-divergence between them can be simplified is:

$$\begin{aligned}
&\text{KL}[q(\mathbf{z}|\mathbf{x}_{\text{in}})||p(\mathbf{z}|\mathbf{x}_{\text{in}})] = \text{KL}[\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)||\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)] \\
=&\frac{1}{2}[\log|\boldsymbol{\Sigma}_k| - \log|\boldsymbol{\Sigma}_x| - z_{\text{dim}} + \text{tr}(\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\Sigma}_x) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}_x)^{\text{T}}\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_x)] \\
=&\frac{1}{2}\left[\text{tr}(\boldsymbol{\Sigma}_x) - \log|\boldsymbol{\Sigma}_x| + \|\boldsymbol{\mu}_x - \boldsymbol{\mu}_k\|^2 - z_{\text{dim}}\right]
\end{aligned}$$

where tr denotes the trace of a square matrix. Since $\boldsymbol{\Sigma}_x$ is a diagonal matrix, $\text{tr}(\boldsymbol{\Sigma}_x)$ is the sum of all elements in $\Sigma(\mathbf{x}_{\text{in}})$, and $\log|\boldsymbol{\Sigma}_x|$ is the sum of the logarithm of elements in $\Sigma(\mathbf{x}_{\text{in}})$.

We can maximize $p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$ by minimizing the reconstruction error $\|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z})\|^2$ and the KL-divergence between $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ and $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ simultaneously. The loss function of FOOD is expressed as follows:

$$L = \|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z})\|^2 + \beta\left(\text{tr}(\boldsymbol{\Sigma}_x) - \log|\boldsymbol{\Sigma}_x| + \|\boldsymbol{\mu}_x - \boldsymbol{\mu}_k\|^2\right)$$

where β is a parameter to balance the tradeoff between reconstruction error and the KL-divergence between $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ and $p(\mathbf{z}|\mathbf{x}_{\text{in}})$.

5.4.2 Classification Algorithm of FOOD

For $\mathbf{x}_{\text{in}}$ belong to class k, FOOD is trained to make $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ close to $p(\mathbf{z}|\mathbf{x}_{\text{in}}) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. Hence, samples $\mathbf{z}$ generated by data from class k are centered around $\boldsymbol{\mu}_k$ and form a cluster, denoted as C_k. Since distributions $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ for different k are designed to be separable, clusters C_k for $k \in \mathcal{Y}$ are apart from each other. These properties enable FOOD to classify the input data $\mathbf{x}_{\text{in}}$ based on the distribution of $\mathbf{z}$.

Although we try to make $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ close to $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ in the training process, $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ can deviate from $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ a lot. We need to recompute the mean vectors $\hat{\boldsymbol{\mu}}_k$ and the covariance matrices $\hat{\boldsymbol{\Sigma}}_k$ for each cluster C_k. We randomly select a portion of ID data points as validation data and estimate $\hat{\boldsymbol{\mu}}_k$ and $\hat{\boldsymbol{\Sigma}}_k$ *based on the validation data instead of the training data*. Although validation data and training data are generated from the same distribution, deep learning models always generate biased results towards training data. Since the validation data are held back during the training, parameters $\hat{\boldsymbol{\mu}}_k$ and $\hat{\boldsymbol{\Sigma}}_k$ evaluated on the validation data are unbiased and are expected to generalize well to ID testing data. Suppose we have N_s samples $\mathbf{z}_{ki}, i = 1, \cdots, N_s$ sampled from $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ for inputs $\mathbf{x}_{\text{in}}$ belong to class k. Parameters $\hat{\boldsymbol{\mu}}_k$ and $\hat{\boldsymbol{\Sigma}}_k$ are estimated as:

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_s}\sum_{i=1}^{N_s}\mathbf{z}_{ki}, \quad \hat{\boldsymbol{\Sigma}}_k = \frac{1}{N_s}\sum_{i=1}^{N_s}(\mathbf{z}_{ki} - \hat{\boldsymbol{\mu}}_k)(\mathbf{z}_{ki} - \hat{\boldsymbol{\mu}}_k)^{\mathrm{T}}.$$

In other words, we expect samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ for $\mathbf{x}_{\text{in}}$ belong to class k follow the distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, but in practice the distribution is close to $\mathcal{N}(\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$. The calibrated Gaussian distribution may introduce different variance scales and covariance among variables in different dimensions. Since the Euclidean distance fails to capture these characteristics properly, we use the Mahalanobis distance, defined as $\sqrt{(\mathbf{z} - \hat{\boldsymbol{\mu}}_k)^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}_k^{-1}(\mathbf{z} - \hat{\boldsymbol{\mu}}_k)}$, to measure the distance between $\mathbf{z}$ and the mean vector $\hat{\boldsymbol{\mu}}_k$. The Mahalanobis distance takes into account the covariance among the variables in calculating distances. Therefore, the problems of scale and correlation are no longer an issue. For each input $\mathbf{x}_{\text{in}}$, the encoder outputs a distribution of $\mathbf{z}$ rather than a single vector $\mathbf{z}$. We define the average value of the squared Mahalanobis distance between $\mathbf{z}$ and $\hat{\boldsymbol{\mu}}_k$ as the distance between $\mathbf{x}_{\text{in}}$ and cluster C_k, and

denote it as $D(\mathbf{x}_{\text{in}}, C_k)$. $D(\mathbf{x}_{\text{in}}, C_k)$ can be explicitly expressed as:

$$\begin{aligned} D(\mathbf{x}_{\text{in}}, C_k) =& \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x}_{\text{in}})}\left[(\mathbf{z}-\hat{\boldsymbol{\mu}}_k)^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}_k^{-1}(\mathbf{z}-\hat{\boldsymbol{\mu}}_k)\right] \\ =& \int (\mathbf{z}-\hat{\boldsymbol{\mu}}_k)^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}_k^{-1}(\mathbf{z}-\hat{\boldsymbol{\mu}}_k)\cdot\frac{1}{(2\pi)^{\frac{z_{\text{dim}}}{2}}|\boldsymbol{\Sigma}_x|^{\frac{1}{2}}}\cdot\exp\left(-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu}_x)^{\mathrm{T}}\boldsymbol{\Sigma}_x^{-1}(\mathbf{z}-\boldsymbol{\mu}_x)\right)d\mathbf{z} \\ =& (\boldsymbol{\mu}_x-\hat{\boldsymbol{\mu}}_k)^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}_k^{-1}(\boldsymbol{\mu}_x-\hat{\boldsymbol{\mu}}_k)+\mathrm{tr}(\hat{\boldsymbol{\Sigma}}_k^{-1}\boldsymbol{\Sigma}_x). \end{aligned}$$

which is the square of Mahalanobis distance between $\boldsymbol{\mu}_x$ and $\hat{\boldsymbol{\mu}}_k$ plus the trace of the matrix $\hat{\boldsymbol{\Sigma}}_k^{-1}\boldsymbol{\Sigma}_x$. *The classification result is chosen to be* $y = \text{argmin}_k\, D(\mathbf{x}_{\text{in}}, C_k), k \in \mathcal{Y}$.

5.4.3 OOD Data Detection Criteria

In section 5.3, we introduced two criteria to detect OOD data and explained their efficacy at a high level. We further provide their formal mathematical details as follows:

(1) *Detecting OOD data based on distances* $D(\mathbf{x}_{\text{in}}, C_k)$. If $\mathbf{x}_{\text{in}}$ is ID data and is predicted to belong to class k, then samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ are in the vicinity of C_k. As a result, the distance $D(\mathbf{x}_{\text{in}}, C_k)$ should be small. If $D(\mathbf{x}_{\text{in}}, C_k)$ is above a threshold τ_k, we regard the input data as OOD. In practice, τ_k are set to different values for $k \in \mathcal{Y}$ and can be estimated using the validation data associated with label k.

(2) *Detecting OOD data based on reconstruction errors.* If $\mathbf{x}_{\text{in}}$ is ID data, then the probability $p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$ should be high. Since $\text{Dec}(\mathbf{z})$ is computed using a deep neural network, $p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$ cannot be derived analytically. Instead, we use Monte Carlo method to estimate the probability value:

$$\begin{aligned} p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}}) =& \int p(\mathbf{x}_{\text{out}}|\mathbf{z},\mathbf{x}_{\text{in}})q(\mathbf{z}|\mathbf{x}_{\text{in}})d\mathbf{z} \approx \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{x}_{\text{out}}|\mathbf{z}_i) \\ \propto& \frac{1}{N}\sum_{i=1}^{N}\exp\left(-\frac{\|\mathbf{x}_{\text{in}}-\text{Dec}(\mathbf{z}_i)\|^2}{2\sigma}\right), \quad \mathbf{z}_i \sim q(\mathbf{z}|\mathbf{x}_{\text{in}}) \end{aligned}$$

Instead of estimating $p(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$, we simplify the computation by calculating the average of reconstruction errors $\|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z}_i)\|^2, i = 1, \cdots, N$ to avoid the estimation of σ. For each class, we define a different threshold e_k. If the average of reconstruction errors of N samples is larger than e_k, then the input data $\mathbf{x}_{\text{in}}$ is considered as OOD. e_k can also be estimated using validation data that belong to class k.

FOOD is able to reduce the failures of OOD data detection by ensuring both criteria are met. First, the encoder maps most forms of OOD data to $q(\mathbf{z}|\mathbf{x}_{\text{in}})$ such that samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ are far from all clusters C_k. As a result, distances $D(\mathbf{x}_{\text{in}}, C_k)$ are large for all $k \in \mathcal{Y}$. But given OOD data $\mathbf{x}_{\text{in}}$, it is possible that $D(\mathbf{x}_{\text{in}}, C_k)$ is small for some k, which indicates that samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}_{\text{in}})$ are in the vicinity of C_k. In this case, $\text{Dec}(\mathbf{z})$ generated by the decoder are similar to class k ID data. Since OOD data are dissimilar to ID data, the reconstruction error is expected to be high, which indicates that such OOD data can be detected by the second criterion. Besides, applying these two criteria does not misclassify ID data that are overlapped between multiple classes as OOD. For such ID data, multiple values of $D(\mathbf{x}_{\text{in}}, C_k)$ could be small. But such property has no impact on OOD data detection.

5.5 Implementation Details of FOOD

5.5.1 Architecture of FOOD

The architecture details of FOOD are summarized in Table 5.1. We select the hyperparameters according to the following rules: (1) The number of downsampling layers in the encoder is equal to the number of upsampling layers in the decoder. (2) Select the model that has the best performance using the validation dataset. (3) If multiple models performed equally well, choose the simplest one.

The input to the encoder is either a 38×100 CWT matrix or a 60×60 density plot. We normalize all entries in the inputs to $[0, 1]$. The encoder contains six convolutional layers. Based on the dimension of the input data, downsampling both width and height by a factor of two for three times so that the height and width range from four to seven in the final convolutional layer is the most suitable choice. We take stride $= 2$ in the convolutional layers for downsampling instead of using 2×2 pooling layers. Such modifications improve performance and reduce the disparity of the performance in multiple training processes. The output of the sixth convolutional layer is flattened and is fully connected (FC) to a hidden layer of dimension 1024. We use Leaky ReLU (Leaky ReLU$(x) = \max\{\alpha x, x\}$, where $\alpha \in (0, 1)$ is a predefined parameter) activation functions with $\alpha = 0.2$ in all convolutional layers and the hidden layer. The output is a vector of size 2×64, where the first 64 elements represent $\boldsymbol{\mu}_x$ and the last 64 elements represent the diagonal elements of $\boldsymbol{\Sigma}_x$. No activation function is applied to the first 64 elements (i.e. $\boldsymbol{\mu}_x$ is a linear combination of neurons in the previous layer) and softplus function ($f(x) = \ln(1 + \exp(x))$) is applied to the last 64 elements so that all diagonal entries in $\boldsymbol{\Sigma}_x$ are above zero. We apply batch normalization [38] to all the convolutional layers, but not to the hidden layer and the output layer.

The input to the decoder is a random variable $\mathbf{z}$ sampled from $q(\mathbf{z}|\mathbf{x}_{\text{in}})$. The first two layers are fully connected layers that extend $\mathbf{z}$ to a vector of dimension $H/8 \times W/8 \times 256$. Then the vector is reshaped to a tensor of dimension $(H/8, W/8, 256)$. Next, we upsample the tensor using deconvolutional filters with size 3×3 and stride 2×2 three times (the output layer is the third deconvolutional layer), and obtain the output with the same dimension as the input. We use the sigmoid function in the output layer and Leaky ReLU with $\alpha = 0.2$ for all other layers. We apply batch normalization to all layers except the output layer.

Researchers may have classifiers in hand that are suitable for their own applications. We assert that it is not difficult to incorporate standard CNN or DNN architectures into FOOD.

Table 5.1: Details of FOOD Architecture

Encoder			
Input: CWT matrix or density plot. Size: $H \times W$, 1 channel			
Layer	Dimension	Stride	Activation Function
Conv	3×3, 32 channels	2×2	Leaky ReLU
Conv	3×3, 32 channels	1×1	Leaky ReLU
Conv	3×3, 64 channels	2×2	Leaky ReLU
Conv	3×3, 128 channels	1×1	Leaky ReLU
Conv	3×3, 256 channels	2×2	Leaky ReLU
Conv	3×3, 256 channels	1×1	Leaky ReLU
FC	1024	—	Leaky ReLU
Output	2×64	—	Linear & Softplus
Decoder			
Input: $\mathbf{z} \sim q(\mathbf{z}\|\mathbf{x}_{\text{in}})$			
Layer	Dimension	Stride	Activation Func
FC	1024	—	Leaky ReLU
FC	$H/8 \times W/8 \times 256$	—	Leaky ReLU
Reshape	$H/8 \times W/8$, 256 channels	—	—
Deconv	$H/4 \times W/4$, 128 channels	2×2	Leaky ReLU
Deconv	$H/2 \times W/2$, 64 channels	2×2	Leaky ReLU
Output	$H \times W$, 1 channel	2×2	Sigmoid

The last two layers of CNN and DNN models are always a fully connected layer followed by the output layer. To build the encoder of FOOD, researchers only need to replace the output layer with a vector of dimension $2z_{\text{dim}}$. The decoder can be built with one fully connected hidden layer followed by three or four deconvolutional layers, which is a widely used structure in generative networks [11, 85]. All parameters need to be retrained, but parameters from existing models can be used as the initial values in the training process.

5.5.2 Parameter Settings and Training Details of FOOD

In this subsection, we first summarize several rules to set the parameters of FOOD. Then we provide some training details.

(1) Due to the curse of dimensionality, distance becomes meaningless in very high dimensional spaces. Besides, the covariance matrix $\hat{\mathbf{\Sigma}}_k$ cannot be estimated accurately if its dimension is too high. On the contrary, z_{dim} should be large enough to preserve essential information for the input data. In our experiments, we choose $z_{\text{dim}} = 64$.

(2) We set the covariance matrices $\mathbf{\Sigma}_k$ to be identity matrices to simplify the computations. Mean vectors are chosen such that the Euclidean distances between $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}_i$ are the same for all $k \neq i$. In our experiments, we choose the $2k-1$ and the $2k$ coordinates of $\boldsymbol{\mu}_k$ to be 5 and other coordinates to be 0. Since the standard deviation of each dimension in $\mathbf{z}$ is 1, separating the mean values to be 5 standard deviations apart is sufficient to distinguish variables from different classes. Increase such values further can significantly increase the training time and is more vulnerable to overfitting.

In the training process, we use Adam optimizer with default parameters recommended in [41]. The batch size is 64, and the learning rate is 5×10^{-4}. In our experiments, we minimize the cross-entropy between $\mathbf{x}_{\text{in}}$ and $\text{Dec}(\mathbf{z})$ instead of $\|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z})\|^2$ because it speeds up the training process and generates outputs with smaller reconstruction errors. We choose β to be 0.1 to balance the cross-entropy loss and the KL divergence. We stop the training process and restore the parameters from previous epochs when we observe a continuous increase in the loss of the validation data.

5.6 Experimental Setup

5.6.1 Experimental Setup for Transmitter Classification

We generate our own dataset to evaluate the performance of FOOD for transmitter classification tasks. We used commercially available SDRs (USRP 2921 and b200 series) as

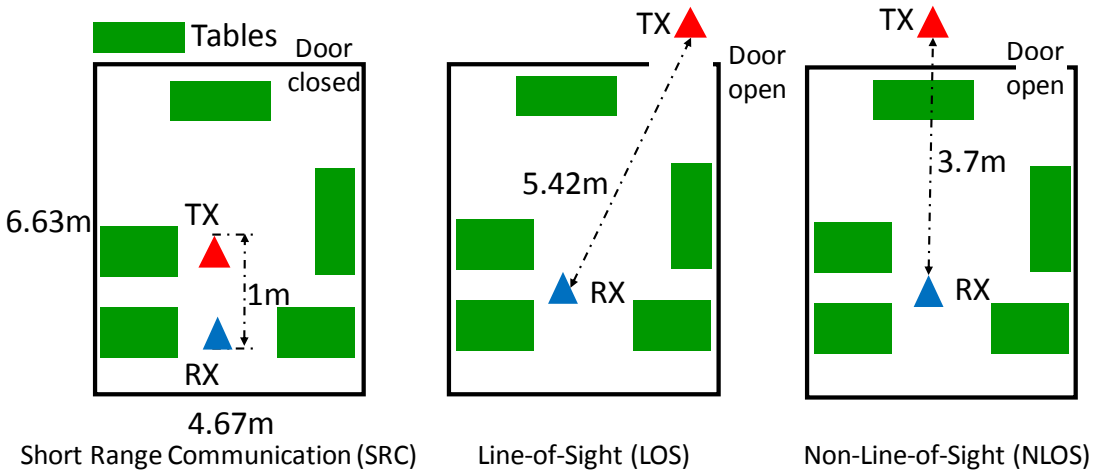


Figure 5.4: Diagrams for different transmission scenarios.

transmitters and the receiver. We considered the following three transmission scenarios in our experiments. The diagrams for these scenarios are illustrated in Figure 5.4.

- Short range communication (SRC): the transmitter and the receiver are only 1 m away without any obstacles between them. The impacts of the channel are minor.
- Line-of-sight (LOS): the transmitter and the receiver are 1.5 meters above the ground and at a distance of 5.42 meters with a clear line-of-sight channel.
- Non-line-of-sight (NLOS): the transmitter and the receiver are 3.7 meters away with a wall between them.

We used four USRP 2921 and four USRP b200 devices in our transmitter classification experiments. For each device, we collected 50K raw IQ data of size 2×100 under each transmission scenario. These samples were subsequently pre-processed using CWT and then used for training and classification.

As stated in section 5.3, FOOD is not designed to detect data generated by different processes but has strong similarities with ID data. In our own dataset, although we have prior

Table 5.2: Summary of experiment setup

Transmitter Classification			
	Data type	Data to be classified	Channel
E1	ID data	3 USRP 2921+3 USRP b200	SRC
	Type 1 OOD	1 USRP 2921+1 USRP b200	SRC
	Type 2 OOD	Same devices as ID data	LOS + NLOS
E2	ID data	3 USRP 2921+3 USRP b200	LOS
	Type 1 OOD	1 USRP 2921+1 USRP b200	LOS
	Type 2 OOD	Same devices as ID data	SRC + NLOS
Modulation Recognition			
E3	ID data	8 modulation schemes	SNR: 22-30 dB
	Type 1 OOD	3 other modulation schemes	SNR: 22-30 dB
	Type 2 OOD	Same schemes as ID data	SNR: 0-8 dB

knowledge that some data are generated under different processes, we do not have ground truth that shows that such data are significantly different from ID data. Besides, defining OOD data using explicit mathematical formulation still remains an open problem. To make OOD data dissimilar to ID data, we chose ID data and OOD data in the following manner.

We take data generated by six transmitters under one scenario listed in Figure 5.4 as ID data. Data generated by the other two transmitters under the same scenario are regarded as Type 1 OOD data. Type 2 OOD data are generated by the same transmitters as ID data but under different channel conditions. It is clear that the NLOS scenario differs significantly from both SRC and LOS scenarios. Our next step is to find a LOS channel such that the generated data are dissimilar to the SRC scenario. We first trained a standard CNN model based on the data generated under the SRC scenario. Then we deliberately changed the positions of the transmitters and observed whether the CNN model can generalize well to the current LOS channel. After observing a sharp drop in accuracy, we stopped moving the transmitters and used the current setup as our LOS transmission scenario.

Based on the previous discussions, we conducted two experiments (also summarized in Table 5.2). In the first experiment (E1), we take data from 3 USRP 2921 devices and 3 b200

devices generated under the SRC scenario as ID data. Data generated by the 4th USRP 2921 device and the 4th b200 device under the SRC scenario are Type 1 OOD data. Data generated by the same devices as ID data but under LOS and NLOS scenarios are Type 2 OOD data. In the second experiment (E2), we use the same devices as E1 to generate both ID and OOD data. But for ID data and Type 1 OOD data, data are generated under the LOS scenario. Type 2 OOD data are generated under the SRC and NLOS scenarios. In both experiments, the ID dataset consists of 300K data points. We randomly choose 70% of ID data as training data, 10% as validation data, and the remaining 20% as testing data. For OOD data, we randomly select 10K data points for each device under each transmission scenario listed in Figure 5.4.

5.6.2 Experimental Setup for Modulation Recognition

We use the public RF dataset [72] that contains both synthetic channel impairments and over-the-air recordings of several modulation schemes. Our experiments focus on classifying the following eight digital modulation schemes: BPSK, QPSK, 8 PSK, 16 PSK, 32 PSK, 16 QAM, 32 QAM, and GMSK. In experiment E3, we take all data generated from SNR values of 22 dB to 30 dB as ID data. Next, data from 8 ASK, 64 QAM, and 256 QAM from SNR values of 22 dB to 30 dB are regarded as Type 1 OOD data. Lastly, data generated from SNR values of 0 dB to 8 dB for the same modulation schemes as ID data are regarded as the Type 2 OOD data. We noticed that data generated from SNR values of 10 dB to 20 dB are very similar to ID data, and the classifier trained using ID data generalizes well to them. Hence, we did not consider data from SNR values of 10 dB to 20 dB as OOD. We note that data below -2 dB were too noisy to be used for classification, so we did not consider them as well. Similar to transmitter classification, we randomly choose 70%, 10%, and 20% data as training, validation, and testing data. Experiment setup is also summarized in Table 5.2.

5.7 Performance Evaluation of FOOD

5.7.1 Classification Accuracy of FOOD

In this subsection, we do not consider the existence of OOD data. We only evaluate the classification accuracy of ID testing data and demonstrate that our new classification method achieves similar performance with standard deep learning-based classification models that use softmax functions in the output layer.

We use the Medium CNN introduced in Chapter 4.5 as a baseline for comparison. Classification accuracy in all experiments are summarized in Table 5.3. The results are evaluated by running the experiments 10 times independently. We can observe that these two models achieve very similar performance. Transmitters can be classified almost perfectly in E1. But as we increase the distance between the transmitter and the receiver, the channel deterioration becomes more significant thereby reducing the accuracy to 83.8% in E2. In E3, the largest source of error is in differentiating 16 PSK and 32 PSK. Such a phenomenon was reported in [73], but our experiments make the classification more challenging because our testing data include data generated under multiple SNR values, while [73] test the accuracy under different SNR values independently.

5.7.2 Impacts of OOD Data

For any input data, a standard classifier simply assigns it to the label that has the highest value in the softmax layer. Therefore, *Type 1 OOD data fools the classifier to classify irrelevant data as existing classes*. In our experiments, a new class of Type 1 OOD data is always being classified as one particular class. For example, more than 95% of the data generated by the 4th USRP 2921 device are classified as being transmitted by one particular

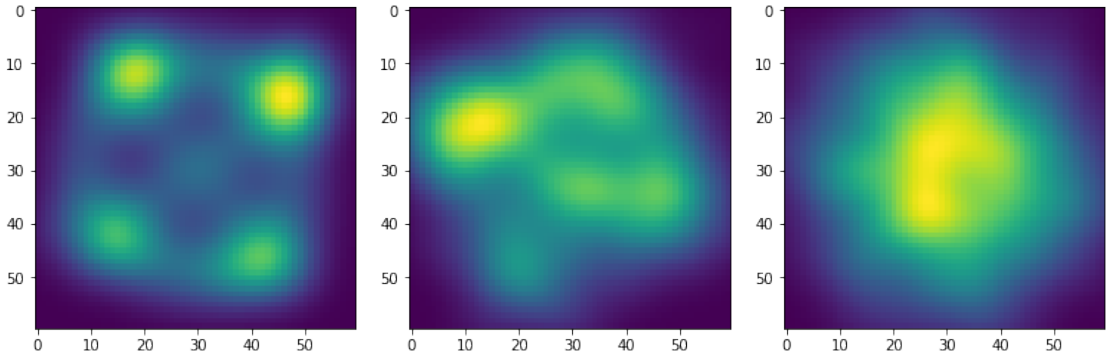
Table 5.3: Comparison of classification accuracy

	Classification accuracy (%)	
Experiment	CNN	FOOD
E1	95.26±0.62	95.07±0.87
E2	83.77±0.66	83.85±1.12
E3	86.59±0.88	86.44±1.40

USRP 2921 transmitter. *Type 2 OOD data tends to generate higher classification errors within existing classes.* A comparison between ID data (SNR: 22 dB) and Type 2 OOD data (SNR: 4 dB and 0 dB) for modulation recognition is illustrated in Figure 5.5. As SNR decreases, the density plot becomes more unstructured. The features extracted from the high SNR density plots may not be useful in classifying low SNR density plots. Although certain Type 2 OOD data can still be classified correctly, there is no performance guarantee. In other words, we do not have prior knowledge of whether the classifier can work reliably when the radio propagation data are transmitted under a different channel condition. For instance, when we train the CNN model using ID data in E1, the classification accuracy for data generated under LOS and NLOS scenarios drops to 62.8% and 40.6%, respectively. Therefore, we assert that the existence of OOD data could decrease the reliability of deep learning-based classification models significantly.

5.7.3 OOD Data Detection of FOOD

For DNN or CNN-based classification algorithms, the most widely used approach to detect OOD data is to use the confidence score [34, 53]. However, we observed some limitations of this approach in our experiments. First, the confidence scores for different classes vary a lot in E3. Several modulation schemes such as BPSK have confidence scores higher than 0.95, while most confidence scores for 16 PSK and 32 PSK are less than 0.6. This suggests that it is difficult to detect OOD data by setting a threshold for confidence scores. Second, more



(a) SNR: 22 dB. (b) SNR: 4 dB. (c) SNR: 0 dB.

Figure 5.5: Density plots for QPSK at different SNR.

than 80% of Type 1 OOD data in E1 and E2 have very similar or even higher confidence scores than ID data. The detection performance cannot be further improved after applying techniques introduced in [53].

In unsupervised learning, OOD testing data can be detected by VAE using reconstruction probability or log-likelihood of the input data. Since log-likelihood has been shown to have significant limitations [67], we compare the performance of FOOD with a standard VAE model that detects OOD data based on reconstruction probability of the input data[1]. The baseline VAE that we compare has the same encoder and decoder architecture with FOOD. But the pre-defined distribution $p(\mathbf{z}|\mathbf{x}_{\text{in}})$ is chosen to be $\mathcal{N}(\mathbf{0}, \mathbf{I})$ in the training process. As explained in section 5.1, the major advantage of FOOD over other VAE models is that FOOD is a unified model to perform classification and OOD data detection simultaneously. Besides, we demonstrate that FOOD outperforms the standard VAE in terms of OOD data detection accuracy.

We adopt three commonly used metrics to measure the effectiveness of FOOD in detecting

[1]In general VAE models, the reconstruction probability is derived from the reconstruction error. In this case, detecting OOD data based on reconstruction probability is equivalent to detecting OOD data based on reconstruction error.

Table 5.4: Metrics for OOD data detection

	OOD type	FPR (95% TPR)		AUROC		AUPR	
		FOOD	VAE	FOOD	VAE	FOOD	VAE
E1	Type 1	5.08	6.12	99.03	98.70	99.11	98.81
	Type 2	0.95	1.23	99.32	99.29	99.40	99.34
E2	Type 1	10.96	14.17	97.82	96.79	98.21	97.35
	Type 2	9.47	12.59	98.14	96.94	98.32	97.79
E3	Type 1	0.48	0.55	99.54	99.52	99.58	99.57
	Type 2	3.37	5.28	99.19	98.96	99.28	99.04

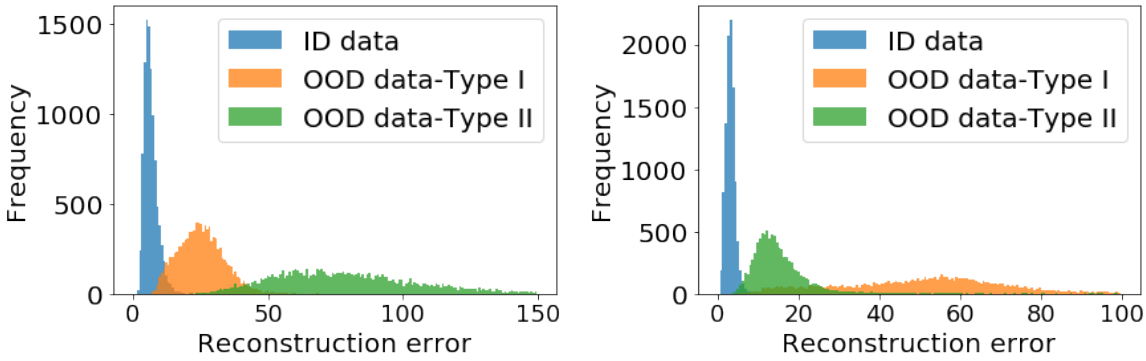
OOD data. In our experiments, *ID data are specified as positives, and OOD data are specified as negatives.* The OOD data detection results for both FOOD and the standard VAE model are summarized in Table 5.4.

1. FPR at 95% TPR: the value of the false positive rate (FPR) when the true positive rate (TPR) is 95%. TPR can be computed as TPR = TP / (TP+FN), where TP and FN denote true positives and false negatives, respectively. FPR can be computed as FPR = FP / (FP+TN), where FP and TN denote false positives and true negatives, respectively. The lower value of this metric, the better.

2. AUROC: area under the receiver operating characteristic curve. A ROC curve depicts the relationship between TPR and FPR. A perfect OOD data detector corresponds to an AUROC score of 100%.

3. AUPR: area under the precision-recall curve. The PR curve is a plot showing the precision = TP / (TP+FP) and recall = TP / (TP+FN) against each other. A perfect OOD data detector corresponds to an AUPR score of 100%.

For input $\mathbf{x}_{\text{in}}$, FOOD first solves $\text{argmin}_k \, D(\mathbf{x}_{\text{in}}, C_k)$, and then verifies whether $D(\mathbf{x}_{\text{in}}, C_k) < \tau_k$ and whether the reconstruction error is less than e_k. These thresholds are user-defined and are determined based on the expected false positive rate and false negative rate. We



(a) Reconstruction errors in E1.

(b) Reconstruction errors in E3.

Figure 5.6: Histogram plots of reconstruction errors.

observed that reconstruction errors are more effective in detecting OOD data than $D(\mathbf{x}_{\text{in}}, C_k)$. Histograms of the reconstruction errors for both ID and OOD data in E1 and E3 are shown in Figure 5.6. The histogram plot for E2 is similar to Figure 5.6a, but with a larger overlap among the histograms associated with ID and OOD data. The reconstruction errors are estimated by averaging $\|\mathbf{x}_{\text{in}} - \text{Dec}(\mathbf{z})\|^2$ for five samples of $\mathbf{z}$. In our experiments, sampling $\mathbf{z}$ for five times is sufficient to estimate the reconstruction error reliably. Since the size of testing data are different for each experiment, we randomly select 10K results for each type of testing data for better visualization and comparison.

By comparing Figure 5.6a and Figure 5.6b, we notice that Type 1 OOD data have smaller reconstruction errors than Type 2 OOD data in E1, but the trend is opposite in E3. This is because transmitter classification relies on subtle PHY layer device features. In our experiments, Type 1 OOD data are generated by devices of the same manufacturer and model series (USRP 2921 and USRP b200 series). The change of channel conditions results in IQ data of more dissimilarities than Type 1 OOD data. But in modulation recognition, changing channel conditions by varying SNR values introduces fewer dissimilarities compare with Type 1 OOD data.

Since reconstruction errors play more important roles in detecting OOD data, thresholds τ_k and e_k for $k \in \mathcal{Y}$ are set as follows. First, we choose a value $\alpha \in [0, 1]$. The value of α is highly correlated with the recall for class k. Then we set τ_k such that when we consider all the class k validation data, the fraction of ID data that are correctly detected as ID is $\alpha^{2/3}$ when we only use $D(\mathbf{x}_{\text{in}}, C_k)$ as the OOD data detection metric. We set e_k such that for all class k validation data, the fraction of ID data that are correctly detected ID is $\alpha^{1/3}$ when we only use reconstruction error as the OOD data detection metric. By varying values of α, we obtain different thresholds τ_k and e_k. We compute the TPR, FPR, precision, and recall (equal to TPR) for the whole testing dataset by varying the value of α in $[0, 1]$ and obtain the results in Table 5.4.

Comparing the results from E1 and E2 in Table 5.4, we observe that when the channel condition deteriorates, it becomes more difficult to detect OOD data accurately. This is because as the channel condition deteriorates, the entropy of ID data becomes higher, which makes ID data more "scattered" in the sample space. Such a phenomenon makes it challenging for FOOD to detect OOD data. Moreover, FOOD outperforms the standard VAE model that solely relies on reconstruction probability to detect OOD data. In a standard VAE, it is possible to learn the structures of the OOD data from the other parts of the ID data. Such OOD data has a high reconstruction probability and cannot be detected effectively. On the contrary, FOOD is able to detect such OOD data by analyzing the corresponding feature representations. Hence, FOOD results in better performance when compared to standard VAE models by taking both distances $D(\mathbf{x}_{\text{in}}, C_k)$ and reconstruction errors into account.

5.8 Chapter Summary

In this chapter, we proposed a novel model called FOOD to detect OOD data in deep learning-based wireless communications applications. Our model overcomes the limitations of previous works by taking an advantage of VAE. We empirically analyzed the performance of FOOD in transmitter classification and modulation recognition under different channel conditions. FOOD performed well on both classification problems, whose input data are processed using completely different signal processing algorithms. Such properties demonstrate the potential of FOOD to be applied to various wireless signal processing techniques with very few modifications.

Although FOOD is proved to perform well in our experimental settings, FOOD has the following limitations. First, FOOD applies Monte Carlo methods to estimate parameters $\hat{\boldsymbol{\mu}}_k$, $\hat{\boldsymbol{\Sigma}}_k$ and the reconstruction errors, which introduce higher model variances. It will be beneficial to derive analytical solutions to estimate these parameters to reduce the model variances. Second, the current analysis does not support FOOD to work for a very large number of classes. In the training stage, we set mean vectors $\boldsymbol{\mu}_k$ such that the Euclidean distances $\|\boldsymbol{\mu}_k - \boldsymbol{\mu}_i\|_2$ are the same for any $k \neq i$. However, we can find at most $d+1$ vectors that satisfy this property in a d-dimension space. Since increasing the dimension in latent space may cause the curse of dimensionality issue, we need to come up with new mechanisms to choose $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ if the number of classes is very large.

Finding the solution to the aforementioned limitations is left as part of our future work. Besides, as far as we know, there is still no mathematical definition of OOD data. Researchers use the term "significantly different from training data" to refer to OOD testing data. But such a description is very ambiguous. In our opinion, it is a very promising research direction to come up with metrics to define OOD quantitatively.

Chapter 6

Detecting Perception Error Attacks in Autonomous Driving

6.1 Introduction

The security and safety of an autonomous driving system highly depend on the reliability of the equipped sensors. However, sensors are very vulnerable to all kinds of attacks. At Black Hat Europe 2015, Petit *et al.* demonstrated the potential danger posed by PEAs by successfully attacking camera and LIDAR systems using cheap commodity hardware [80]. Later on, Yan *et al.* demonstrated that the sensors on a Tesla Model S can be attacked so that the vehicle's autonomous driving system fails to detect certain objects [108]. Cao *et al.* demonstrated that it is feasible to launch LIDAR spoofing attacks to make Baidu Apollo platform falsely detect nonexistent objects [7].

The less-than-perfect reliability of the object detection and scene recognition algorithms employed by autonomous vehicles also poses a serious risk to safety [6]. Such algorithm failures may result in serious accidents. In 2016, a Tesla driver was killed due to a failure of the autopilot system. The car's sensor data analyzing algorithm failed to detect the presence of a large white 18-wheel truck crossing the highway [32], partially due to the backdrop of a bright spring sky. In March 2018, an Uber self-driving car struck and killed a woman while it was tested in autonomous mode during the night [61].

Multiple sensor fusion algorithms have been proposed to help the system better "understand" the surrounding environment. Due to the insufficient reliability of image processing algorithms, cameras are always used in conjunction with other types of sensors, such as LIDAR. The performance of object detection algorithms can be improved if LIDAR data and camera image data are fused to generate the detection results [2, 12, 37, 103]. Increasing the precision of object classification and object tracking algorithms by fusing multiple types of sensors are proposed in [10, 13, 52, 83]. However, all these works implicitly make an assumption that all sensor data are trustworthy. This assumption creates a vulnerability that can be exploited by PEAs. Sensor fusion algorithms can produce erroneous outputs if some of the sensory data have been corrupted by PEAs. Hence, it is essential to detect PEAs and eliminate corrupted sensor data before fusing them together.

In this chapter, we discuss details of LIFE, a scheme to detect PEAs in autonomous vehicles. As its name implies, the proposed scheme specifically focuses on the fusion of LIDAR and image data, which are most commonly employed by autonomous vehicle manufacturers (e.g., KITTI [28], Ford [74], Waymo [104]) as the primary sensors, to detect instances of PEAs. Besides autonomous driving, LIFE can also be applied to other applications that use stereo cameras and LIDAR, such as robots used for outdoor navigation [74], robots used indoors [47], and Unmanned Aerial Vehicles (UAV) [58].

LIFE detects PEAs by detecting changes in the correlation between LIDAR data and image data. It also considers the temporal correlation between the data points within a sequential dataset. Different types of sensors generate different sensory data in completely different formats. However, the data points of the different sensory datasets manifest a high level of correlation when the different sensors capture the stimuli from the same object. We assert that LIFE can successfully detect most forms of PEAs by detecting a change in the correlation induced by the attacks. To circumvent LIFE—or a similar method that fuses two

or more types of sensory data—an attacker would need to attack multiple types of sensors in such a way that the correlation between the relevant data points is maintained. Our results indicate that such kinds of attacks are very difficult, if not impossible, to carry out in real-world scenarios. In essence, LIFE exploits the *diversity* of the different sensory data as well as the *correlation* between them to detect PEAs.

The idea of using redundant sensors or fusing different types of sensors to increase reliability is not new. However, LIFE is different from the prior art in terms of the following aspects: (1) The primary objective of existing LIDAR and camera fusion algorithms is to increase perception or detection accuracy. Such algorithms are not necessarily designed to detect perception errors or instances of PEAs. On the contrary, LIFE was specifically designed to detect PEAs by employing novel sensor fusion algorithms. (2) The existing schemes [80, 107, 108] that are designed to detect perception errors utilize multiple redundant sensors *of the same type*. On the other hand, LIFE takes advantage of the *diversity of the different types of sensors* employed by a typical autonomous vehicle to detect PEAs. As we will explain in sections 6.5 and 6.6.3, LIFE's approach is more effective in detecting PEAs; and (3) Existing attack resilient sensor fusion schemes [25, 39, 93, 102] use statistical models to characterize the metrics or features obtained from sensor data. These schemes detect erroneous sensor data by solving an optimization problem. This approach is not possible when fusing LIDAR and camera data because the data dimensionality is too high and features are too complicated to be expressed in an explicit statistical model. In contrast, LIFE employs machine learning-based signal processing algorithms to correlate the LIDAR and camera data streams that have high dimensionality and high complexity.

The contribution of this work is summarized below.

- We propose a novel scheme, which we refer to as LIFE, for detecting PEAs targeting autonomous vehicles. LIFE takes advantage of the diversity of the different types of

sensors to detect PEAs without the need for redundant sensors.

- In certain situations, LIFE can also enhance the reliability of autonomous driving systems by detecting instances in which object detection algorithms make mistakes.
- LIFE can be readily integrated into existing autonomous systems without requiring any additional computation hardware.
- The performance of LIFE has been evaluated extensively using the well-known KITTI dataset [28], which is a definitive dataset of an autonomous vehicle's sensory data.

6.2 Related Work

Camera, LIDAR, radar, etc., all have high feasibility of being attacked without requiring physical access [79]. Some intelligent remote attacks such as spoofing, using materials to absorb signals, are almost impossible to be detected by systems without analyzing the correlation among multiple types of sensors. Recent literature has demonstrated that remote attacks on autonomous driving sensors are feasible in real-world experiments. Petit *et al.* attacked the camera and LIDAR of a target vehicle using commodity hardware costing less than $60 [80]. Shin *et al.* extended Petit's work by making illusions appear closer than the location of the spoofer [92]. In addition, they demonstrated the feasibility of saturation attacks, which can completely incapacitate a LIDAR from sensing a certain direction. Cao *et al.* [7] explored the feasibility of strategically generating and placing spoofed LIDAR points to fool the machine learning algorithm that processes the LIDAR data. The researchers were able to achieve an attack success rate of approximately 75% when the attacks were performed against the Baidu Apollo LIDAR perception module. Yan *et al.* performed blinding attacks on camera as well as jamming and spoofing attacks on radar and ultrasonic sensors on a

Tesla S automobile [107, 108]. Furthermore, Ethernet connections or inter-vehicle networks [55, 66] provide more feasibility to hack sensor data [76, 88].

Several methods to detect PEAs have been proposed in [39, 80, 92, 107, 108], including (1) adding more redundant sensors, (2) using inter-vehicle communications to compare sensor measurements, (3) relying on other sensors to detect attacks. However, these proposed methods are likely to be ineffective in detecting PEAs. Although employing multiple sensors of the same type can increase resilience to random faults, it is not effective in defending against intentional attacks, such as PEA, that exploit vulnerabilities of a certain type of sensor apparatus. Using inter-vehicle communications to compare sensor measurements is also an ineffective strategy since this requires the vehicles to have V2V (vehicle to vehicle) communications capability and be located within other vehicles' communication range.

Given the limitations of the other methods, we claim that the most practical approach to detect PEAs is to employ a combination of heterogeneous sensors. Similar ideas were proposed in the literature [7, 92, 108], but no techniques that instantiate such ideas were proposed. LIFE is a concrete realization of the idea that takes advantage of the *diversity of the different types of sensors* employed by a typical autonomous vehicle to detect PEAs.

6.3 Threat Model

6.3.1 PEAs Targeting LIDAR and Camera

A LIDAR is composed of multiple laser transceivers and a rotary system for scanning, as shown in Figure 6.1. Surround-view can be acquired by rotating the lasers and transceivers periodically. A LIDAR detects an object as follows. First, a LIDAR emits laser pulses while spinning. When the emitted pulses hit an object, laser energy reflects back to the LIDAR.

LIDAR can calculate the distance to the object from the elapsed time and the speed of light. The direction of the object can be derived from the rotation angle of its spin. LIDAR can form a *point cloud* by aggregating all the measured points, of which each point's coordinate is the relative position to the LIDAR origin. An example of point cloud is shown in Figure 6.3a. An adversary can attack a LIDAR system by injecting fake echoes or by preventing it from receiving reflected echoes. Well-known attacks against LIDAR are described below:

Spoofing attack: Attackers can inject fake echoes to cause a LIDAR to detect nonexisting objects falsely. The attack method is shown in Figure 6.1. A photodiode is used to synchronize with the victim LIDAR. Then the delay component triggers the attack laser after a certain amount of time to inject fake points in the following LIDAR detection cycles. Up to date, the most effective LIDAR spoofing attack was presented in [7]. Fake points can be generated at all the vertical viewing angles and an 8° horizontal angle at a distance of greater than 10 meters. Approximately 100 fake points can be generated, and 60 of them targeting the center 8-10 vertical LIDAR lasers can be stably spoofed. More spoofed points can be generated using more advanced equipment or multiple devices. Spoofing attacks in [7] can also fool the Baidu Apollo perception module [3] into detecting a cluster of spoofed points as an actual object by carefully controlling the positions of the spoofed points. The victim autonomous vehicle dropped its speed from 43 km/h to 0 km/h within one second, which could possibly cause an accident.

Saturation attack [92]: LIDAR is a type of transducer that converts laser pulse intensity into electrical signals. When the input signal energy level is too high, LIDAR will enter the saturation region. Increasing the input energy level will cause almost no change in output. By illuminating a LIDAR with a strong light of the same wavelength as the LIDAR, one can cause the LIDAR to enter its saturation region, and hence prevent it from detecting reflected echoes. As a consequence, reflected echoes are concealed. To launch a saturation attack, an

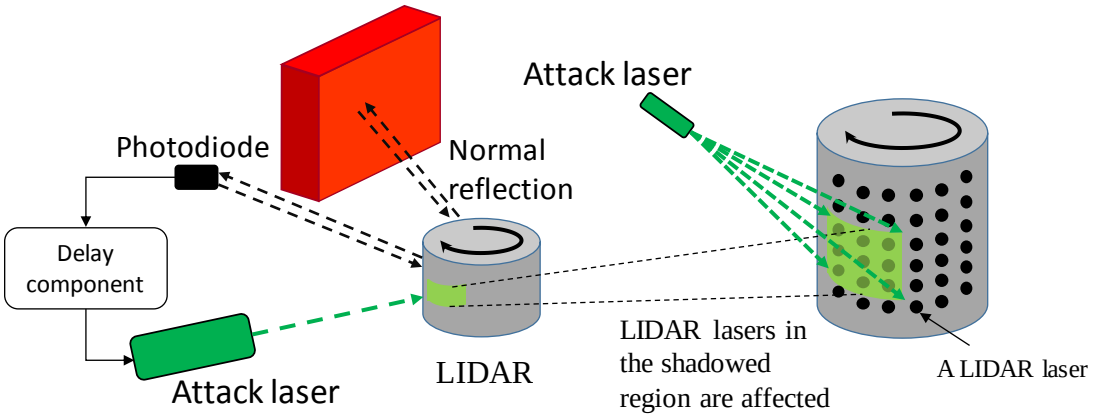


Figure 6.1: Diagram illustrating how to launch PEAs towards LIDAR.

attacker only needs to shoot a laser with the same wavelength towards the victim LIDAR. Experimental results show that objects along certain directions can be completely concealed with a strong light source.

A stereo camera has two lenses with a separate image sensor frame for each lens. The outputs are two *stereo images* with slightly different vision angles, as shown in Figure 6.3b. The most common attack targeting a camera is the **blinding attack** [80, 108]. This attack causes cameras to fail to capture images by shooting light towards camera or objects. Experiments in [108] show that shooting LED light and laser 15° to the axis perpendicular to a camera lens can lead to the camera's complete blindness for three seconds. Moreover, aiming a LED light at an angle of 45° towards an object can conceal the object from a camera.

6.3.2 Attack Model

Based on the experimental results in [7, 80, 92, 108], we consider the following attack model:

Attack scenarios: The attacker can either hide the equipment along the roadside (e.g., hide behind a traffic sign) to attack incoming autonomous vehicles, or drive a vehicle equipped

with devices that can shoot laser pulses to attack the nearby victim's LIDAR and camera. The attacker's goal is to launch PEAs to alter the vehicle's driving decisions and potentially cause serious accidents.

Attacker's capability: We assume that an attacker is able to either launch remote attacks on LIDAR and cameras using the aforementioned equipment or physically attack the sensors in order to cause them to malfunction. However, the attacker has the following constraints to launch PEAs. (1) The attacker does not have access to the data processing system located within the vehicle. (2) Even though an attacker can attack a LIDAR and a camera simultaneously, he/she is not able to maintain the correlation between these two types of sensory data. Maintaining such a correlation requires knowledge of the precise positions of the sensors and the calibration metrics of the victim vehicle. For example, when projecting LIDAR points onto camera images, move the camera by 5 mm can cause tens of pixels projection differences. (3) We assume that the effect of an attack is instantaneous. To attack a camera, an attacker modifies the luminance of a certain region of pixels instantly. To attack a LIDAR, an attacker modifies multiple LIDAR points simultaneously. It may be technically feasible to launch PEAs in a very gradual manner, such as slowly increasing the intensity of the light to attack cameras, or inject very few spoofed LIDAR points over a certain time duration. However, it requires extremely sophisticated control over multiple attack lasers. As far as we know, no research has demonstrated launching PEAs in such a gradual manner. So in our model, we assume the attackers do not have the capability to carry such attacks.

LIFE is designed to detect PEAs targeting sensors rather than attacks targeting sensory data processing algorithms. Prior researches have demonstrated that physical adversarial attacks can fool the perception algorithms. For example, a stop sign can be mistakenly recognized as a speed limit sign by sticking black and white papers [24]. In addition, it is possible to use 3D

printers to create adversarial objects that cannot be detected by LIDAR-based perception algorithms [8]. Under these scenarios, sensors still capture the correct stimuli. Perception failures are caused by the unreliability of perception algorithms. Therefore, LIFE should conclude that both sensors are working properly. Detecting physical adversarial attacks targeting specific perception algorithms is beyond the scope of this paper.

6.4 Overview of LIFE

In this section, we first give an overview of the proposed LIFE system. Then we give a brief introduction to the KITTI dataset, which is used to evaluate the performance of LIFE. LIFE detects PEAs by analyzing the correlation between LIDAR and stereo camera data. For each detected inconsistency, LIFE determines which sensor's data is reliable and which one can be considered anomalous. In summary, LIFE carries out the following two core functionalities:

1. Consistency checking: LIFE detects inconsistencies by correlating LIDAR and camera data.

2. Sensor reliability evaluation: for each detected inconsistency, LIFE determines whether it is caused by PEA and Figures out which sensor is under attack.

Besides determining whether LIDAR or camera is under PEA, LIFE also points out the locations of data points that are affected by PEAs (i.e., positions of spoofed or concealed LIDAR points, regions of problematic image pixels). LIFE outputs all the detected inconsistencies caused by PEAs and provides the corresponding sensor reliability results. Inconsistencies caused by sensors' inherent limitations or errors in object detection algorithms rather than PEAs are ignored by LIFE. A flowchart of LIFE is illustrated in Figure 6.2. A detailed illustration of LIFE procedures is shown in Figure 6.3.

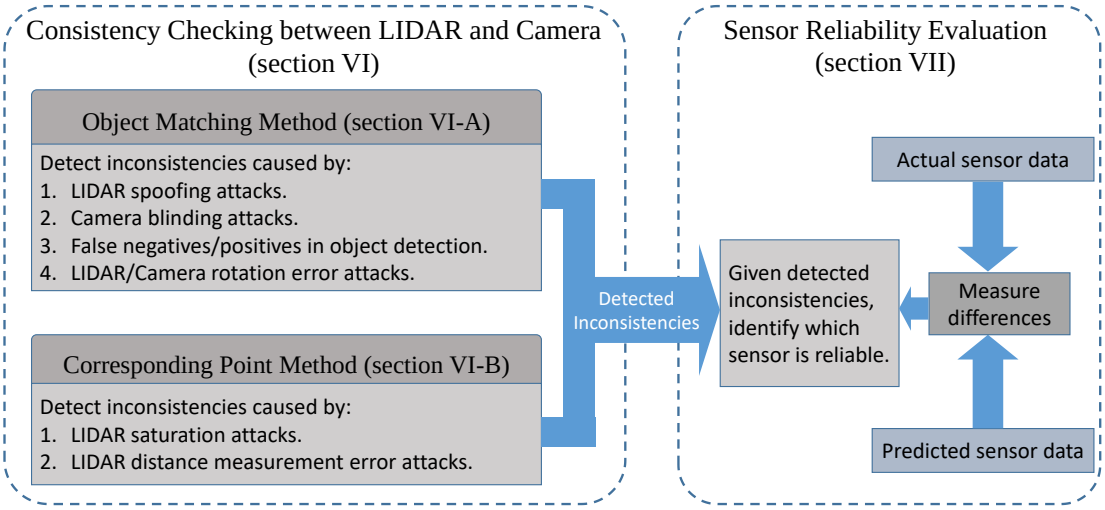


Figure 6.2: Flowchart of LIFE.

6.4.1 Consistency Checking Methods

We use the following two ideas to correlate LIDAR data that senses objects in 3D position and camera data that senses information on 2D image planes. On the one hand, we can project 3D LIDAR data points onto 2D image planes, which corresponds to the object matching method. It is effective in detecting inconsistencies caused by LIDAR spoofing attack, camera blinding attack, object detection errors, etc. On the other hand, we can calculate the position of a 3D LIDAR data point if we know its projected stereo 2D image coordinates, which corresponds to the corresponding point method. It can detect inconsistencies caused by LIDAR saturation attacks and LIDAR distance measurement errors.

Object matching method: The overall idea is to project 3D LIDAR points onto images, then check whether objects detected from LIDAR and images match each other. The procedures are illustrated in Figure 6.3a. It can be summarized in the following steps:

1. Extract objects from LIDAR point cloud. It is composed of the following substeps:
   - Remove data points reflected by the ground. We will call them *ground points* in the

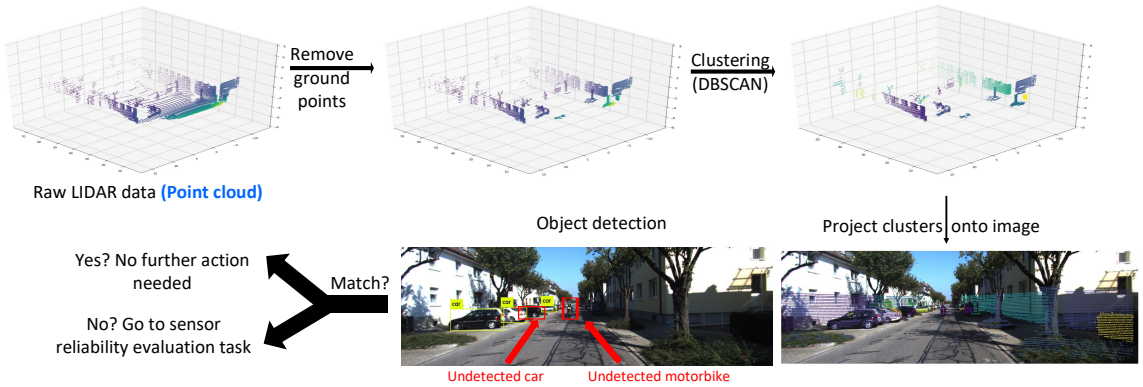
rest of this paper, and we will call the remaining points *aboveground points*.

- Apply DBSCAN (density-based spatial clustering of applications with noise) clustering algorithm [23] to the aboveground points. Each cluster represents an object extracted from LIDAR data.
- Project each cluster onto the image, and record the projected cluster positions.

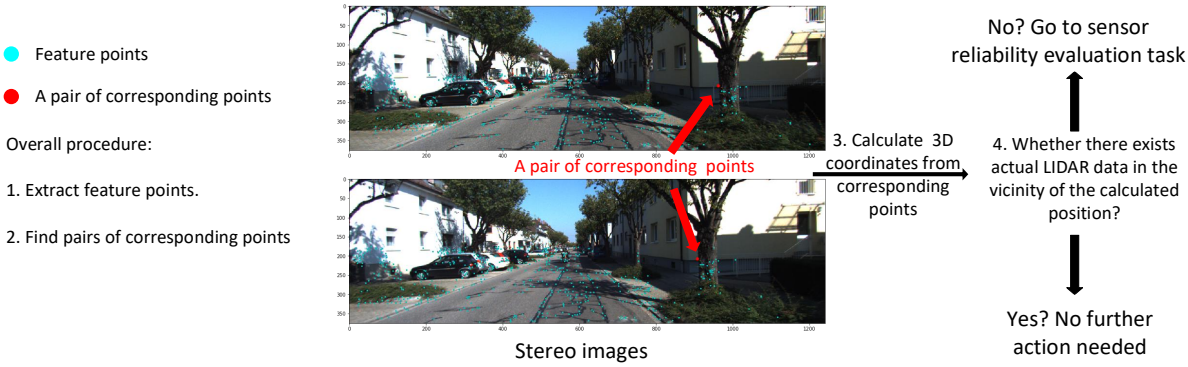
2. Run an object detection algorithm on camera images.
3. Check whether the positions of projected LIDAR clusters match the positions of detected objects on images.

Corresponding point method: The overall idea is to check whether distance information obtained from LIDAR is consistent with distance information obtained from two stereo camera images. The procedures are illustrated in Figure 6.3b. If two points on two stereo images are projected from the same 3D world point, then we call them *a pair of corresponding points*, or a *corresponding pair*, as shown in Figure 6.3b. Given the coordinates of a pair of corresponding points, we can calculate the corresponding 3D position and check whether there exist LIDAR data points in the vicinal region. If no LIDAR points are found nearby, then an inconsistent instance is detected. This method can be summarized as follows:

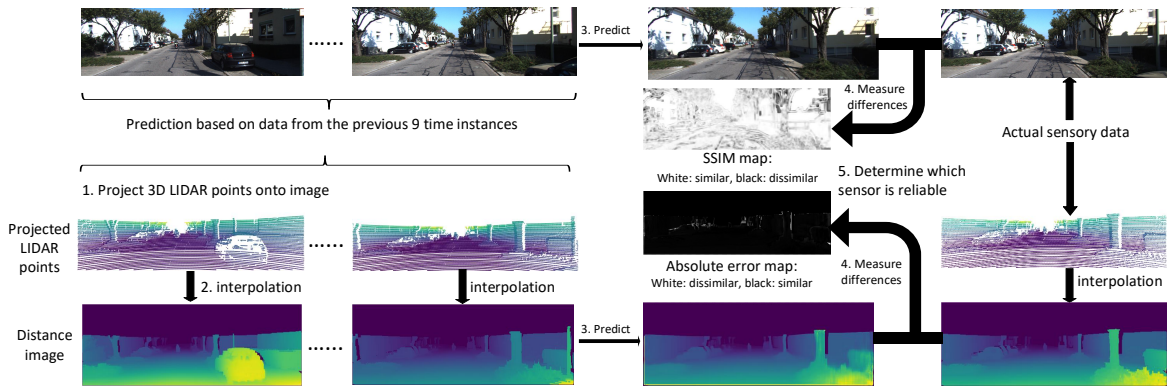
1. Apply scale-invariant feature transform (SIFT) descriptor [57] to find corresponding pairs from stereo images.
2. Compute the 3D position of the point that projects to these two stereo image points.
3. Remove wrongly formed corresponding pairs.
4. Check whether there exist actual sensory LIDAR data points in the vicinity of the calculated 3D position.



(a) Flowchart of object matching method. The idea is to project 3D LIDAR data points onto 2D image plane to check consistencies.



(b) Flowchart of corresponding point method. The idea is to calculate 3D coordinates from 2D stereo image points to check consistencies.



(c) Flowchart of sensor reliability evaluation method. Reliability evaluation is based on differences between predicted data and actual data.

Figure 6.3: Illustration of LIFE procedures using KITTI data.

6.4.2 Sensor Reliability Evaluation

Consistency checking methods can detect the inconsistencies caused by PEAs, but cannot determine which sensor is under PEAs. Identifying unreliable sensors is accomplished by the sensor reliability evaluation method. The procedures are illustrated in Figure 6.3c. The evaluation results are based on measuring the difference between the predicted sensor data and the actual sensory data. At each time instance, we can use preceding sensor data to predict the current data and check how large the difference between the prediction and actual measurements. In this work, we use the deep learning framework described in [56] to predict camera images. However, LIDAR data is very sparse 3D data. Directly inferring LIDAR data in the 3D coordinate system requires a tremendous amount of computation resources. Also, the exact locations of LIDAR points are not predictable. To overcome these challenges, we construct *distance image* for LIDAR data at each time instance, and predict the distance image instead. A distance image is an image whose size is the same as the camera image, and each pixel value represents the distance to LIDAR coordinate origin. Distance image can be constructed by projecting LIDAR data onto the camera image plane and interpolating. The sensor reliability evaluation task is carried out in the following steps:

1. Construct a distance image at each time instance.

2. Predict distance image and camera image at the current time instance based on data in previous time instances.

3. For each detected inconsistency, justify which sensor is more reliable based on the differences between the predicted data and actual sensory data.

6.4.3 Introduction to the KITTI Dataset

LIFE is evaluated using one of the world's most popular autonomous driving dataset, the KITTI Dataset [28]. It serves as a standard dataset for benchmarking autonomous driving tasks, such as object detection, pedestrian, vehicle tracking, etc. The KITTI test vehicle is equipped with a stereo camera with two lenses and a rotating LIDAR. The camera is *synchronized* at about 10 Hz with respect to the LIDAR. The rectified camera image resolution is roughly 1250×370 pixels. The equipped LIDAR has a 360° horizontal field of view, but we are only interested in the data points that can be projected onto camera images. Each LIDAR point uses a 3-dimension vector to represent x, y, z coordinates (corresponding to forward, left, and up direction, unit: meter) in the LIDAR coordinate system. Examples of LIDAR point cloud and stereo camera images at one time instance are shown in Figure 6.3a and Figure 6.3b, respectively.

6.5 Consistency Checking in LIFE

6.5.1 Object Matching Method

Remove ground points: Since object information is only contained in aboveground points, we need to remove ground points first. It can be challenging due to the following two reasons: (1) Ground can be bumpy or sloped; (2) The further the distance to the vehicle, the sparser the LIDAR points are. Simply setting a threshold and regarding LIDAR points lower than the threshold as ground points is infeasible. Here we will introduce our heuristic grid-based algorithm, which is simple but very effective. It consists of the following steps:

- Partition all LIDAR points into $0.5m \times 0.5m$ grids based on their x,y coordinates.

- Among all points in the ith row of the grids, find the minimum value of z, denote it as z_i. If $z_i - z_{i-1} > 0.5$, it indicates that there is no ground point in the ith row, hence set $z_i = z_{i-1}$.
- Regard LIDAR points in the same row with height 0.5 m higher as aboveground points.
- For the remaining points in each grid, if the range of z coordinates is less than 0.2, then all the points in the grid are ground points. Otherwise, regard the points in the lowest 0.2 meters as ground points and others as aboveground points.

LIDAR points clustering: We use DBSCAN, a density based clustering algorithm [23], to cluster aboveground LIDAR points. It can determine the number of clusters automatically. Besides, clusters are formed based on the density of data points without prior statistics distribution assumptions. Both features are beneficial in processing the aboveground LIDAR data points. In our experiments performed on the KITTI dataset, we use $\epsilon = 0.7m$, and MinPts=20. Each cluster represents a detected object from LIDAR points.

Match detected objects: At each time instance, the positions of objects on the image plane can be obtained either from camera images by running object detection algorithms, or from LIDAR data by projecting LIDAR clusters onto the image. We can check the consistency by comparing whether object boundaries from LIDAR and camera are roughly the same. A rectangle is used to represent object boundaries in both algorithms. We use Intersection over Union (IOU) to measure the closeness between detected objects from image and LIDAR. Ideally, real-world objects and LIDAR data clusters form one to one correspondences. But sometimes LIDAR points reflected by different objects may be aggregated into the same cluster. To deal with this issue, we propose Intersection over Minimum (IOM), defined as the union over the minimum area of the two rectangles formed by the LIDAR cluster and object detector, as an additional metric to measure how well objects extracted from LIDAR

match objects obtained from images. The matching method is summarized in Algorithm 1.

The 2D positions of the unmatched objects from images, 3D positions of unmatched LIDAR clusters, and their projected 2D positions are served as the inputs of sensor reliability evaluation step. Unmatched objects are either due to errors in object detection algorithms or due to PEAs. These two scenarios can be distinguished in the sensor reliability evaluation task. In the state-of-art object detection algorithms, there always exist false positives (a non-existing object is falsely detected as a real object) and false negatives (the failure to detect an existing object). These errors are due to the failures of detection algorithms rather than attacks targeting sensors. The sensory data still maintain sequential correlations. Therefore, these object detection errors will not be recognized as PEA instances in the sensor reliability evaluation step. On the contrary, if the inconsistencies are caused by PEAs such as spoofed LIDAR points, the sequential correlation among the sensory data is broken. Such PEAs can be detected in the sensor reliability evaluation step.

Algorithm 1 Object Matching Algorithm

Input: Top left and bottom right corner coordinates for detected objects on images (list_image) and projected LIDAR clusters (list_lidar).
Output: Classify detected objects as matched or unmatched.
Sort list_image based on the occupied area in descending order.
for each object in list_image **do**
    Calculate IOU with all elements in list_lidar, find the maximum.
    if the maximum IOU is larger than 0.5 **then**
        Mark the two objects in list_image and list_lidar as matched.
        Delete both objects from the lists.
    else
        Calculate IOM with all entries in list_lidar, find the maximum.
        if the maximum IOM is larger than 0.9 **then**
            Delete the matched object from list_image.
            Mark the object in list_lidar as matched.
        end if
    end if
end for
Label all remaining objects in list_image and list_lidar as unmatched.

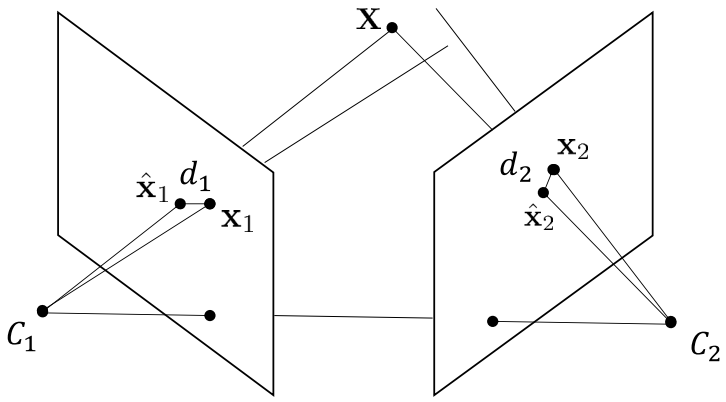


Figure 6.4: Find the 3D point by minimizing the reconstruction error.

6.5.2 Corresponding Point Method

Identify corresponding points: Calculating 3D position from a pair of corresponding points is very sensitive to image coordinate measurement errors. This requires us to localize corresponding point positions accurately. As a result, we need first to identify *feature points* that are significantly distinguishable from other points and use feature points to form corresponding pairs.

Here we use SIFT descriptor [57] to extract feature points. SIFT descriptor is invariant to translation, rotation, and scaling. It uses a 128-dimensional vector to represent each point. The feature vector is calculated based on gradient magnitude and orientations. Points located on weak edges and backgrounds will be discarded by setting some predefined thresholds. After that, corresponding pairs can be formed based on the Euclidean distance of feature vectors using the nearest neighbor method. Corresponding point examples are shown on the stereo images in Figure 6.3b.

Calculate 3D position from corresponding points: We derive a closed-form solution to find the 3D position of a point given its projected coordinates on two stereo camera images. Suppose the 3D Cartesian coordinate of the data point is $\mathbf{x} = [x, y, z]^{\mathrm{T}}$, and its homogeneous

coordinate is $\mathbf{X} = [x, y, z, 1]^{\mathrm{T}} = [\mathbf{x}^{\mathrm{T}}, 1]^{\mathrm{T}}$. Given Cartesian image coordinates of a pair of corresponding points (u_1, v_1), (u_2, v_2) and calibration matrices $\mathbf{P}_1, \mathbf{P}_2$, we want to find $\mathbf{x}$.

We choose to use homogeneous image 2D coordinates $\mathbf{x}_1 = [u_1, v_1, 1]^{\mathrm{T}}$ and $\mathbf{x}_2 = [u_2, v_2, 1]^{\mathrm{T}}$. Without measurement error in both 3D and 2D coordinates, we should have:

$$\mathbf{P}_1\mathbf{X} = k_1\mathbf{x}_1, \quad \mathbf{P}_2\mathbf{X} = k_2\mathbf{x}_2, \tag{6.1}$$

in which k_1 and k_2 are scaling factors to make the last coordinate of $\mathbf{x}_1, \mathbf{x}_2$ equal to 1 (note that $k_1\mathbf{x}$ and $\mathbf{x}$ represent the same point in homogeneous coordinate). However, due to numerical and coordinate measurement errors, it is impossible to find $\mathbf{X}$ to satisfy eq. (6.1) . Instead, we try to find a point $\mathbf{X}$ with its projected positions $\hat{\mathbf{x}}_1 = \mathbf{P}_1\mathbf{X}$ and $\hat{\mathbf{x}}_2 = \mathbf{P}_2\mathbf{X}$, such that the reprojection error $d_1 = \|\mathbf{x}_1 - \hat{\mathbf{x}}_1\|$ and $d_2 = \|\mathbf{x}_2 - \hat{\mathbf{x}}_2\|$ can be minimized, as shown in Figure 6.4. C, C_1 and C_2 are camera centers. $\mathbf{X}$ is a 3D point, $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}$ are projected image points. Hence, we can formalize an optimization problem as:

$$\text{Minimize} \quad \|\mathbf{P}_1\mathbf{X} - k_1\mathbf{x}_1\|^2 + \|\mathbf{P}_2\mathbf{X} - k_2\mathbf{x}_2\|^2 . \tag{6.2}$$

with variables $\mathbf{X}$ (or $\mathbf{x}$), k_1, k_2. Denote $\mathbf{P}_{ij}$ as the jth ($j = 1, 2, 3, 4$) column of matrix $\mathbf{P}_i$ ($i = 1, 2$). Then we have:

$$\mathbf{P}_1\mathbf{X} - k_1\mathbf{x}_1 = [\mathbf{P}_{11}, \mathbf{P}_{12}, \mathbf{P}_{13}, -\mathbf{x}_1, \mathbf{0}] \cdot \left[\mathbf{x}^{\mathrm{T}}, k_1, k_2\right]^{\mathrm{T}} + \mathbf{P}_{14},$$
$$\mathbf{P}_2\mathbf{X} - k_2\mathbf{x}_2 = [\mathbf{P}_{21}, \mathbf{P}_{22}, \mathbf{P}_{23}, \mathbf{0}, -\mathbf{x}_2] \cdot \left[\mathbf{x}^{\mathrm{T}}, k_1, k_2\right]^{\mathrm{T}} + \mathbf{P}_{24}.$$

For simplicity, define

$$\mathbf{W}_1 = [\mathbf{P}_{11}, \mathbf{P}_{12}, \mathbf{P}_{13}, -\mathbf{x}_1, \mathbf{0}], \mathbf{W}_2 = [\mathbf{P}_{21}, \mathbf{P}_{22}, \mathbf{P}_{23}, \mathbf{0}, -\mathbf{x}_2], \quad \mathbf{X}^+ = [\mathbf{x}^{\mathrm{T}}, k_1, k_2]^{\mathrm{T}}.$$

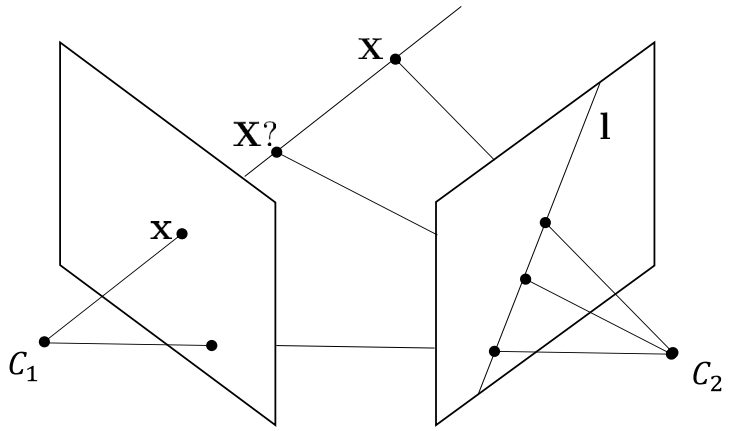


Figure 6.5: Geometry explanation of fundamental matrix.

Then the optimization problem (6.2) is changed to:

$$\text{Minimize } \|\mathbf{W}_1\mathbf{X}^+ + \mathbf{P}_{14}\|^2 + \|\mathbf{W}_2\mathbf{X}^+ + \mathbf{P}_{24}\|^2.$$

The optimal solution is:

$$\mathbf{X}^+ = -(\mathbf{W}_1^{\mathrm{T}}\mathbf{W}_1 + \mathbf{W}_2^{\mathrm{T}}\mathbf{W}_2)^{-1} \cdot (\mathbf{W}_1^{\mathrm{T}}\mathbf{P}_{14} + \mathbf{W}_2^{\mathrm{T}}\mathbf{P}_{24}).$$

Note that the last two dimensions of $\mathbf{X}^+$ are scaling factors that can be simply discarded. The first three dimensions are the needed 3D coordinates in Cartesian coordinate system.

Remove wrongly formed corresponding pairs: In practice, there exist some pairs of points with similar SIFT features, but are actually projected from different points. We need to eliminate the wrongly formed corresponding pairs.

The geometry explanation of fundamental matrix is demonstrated in Figure. 6.5. C_1 and C_2 are camera centers. $\mathbf{X}$ is a 3D point and $\mathbf{x}$ is the projected image point. As shown in Figure 6.5, a 3D point projected onto $\mathbf{x}$ must be located on the ray connecting $\mathbf{x}$ and the

center of the left stereo camera C_1. This ray is projected on line $\mathbf{l}$ on the right image plane. That means a corresponding point of $\mathbf{x}$ must lie on the line $\mathbf{l}$. Suppose $\mathbf{x}$ is an imaged point on the left image plane, and $\mathbf{x}'$ is a corresponding point of $\mathbf{x}$ on the right image plane. The aforementioned relationship can be characterized by the fundamental matrix $\mathbf{F}$ as $\mathbf{x}'^{\mathrm{T}}\mathbf{F}\mathbf{x} = 0$. It should be noted that $\mathbf{x}^{\mathrm{T}}\mathbf{F}\mathbf{x}'$ is not necessarily equal to 0. Given $\mathbf{P}_1$ and $\mathbf{P}_2$, the calibration matrix of the left stereo camera and the right stereo camera, the fundamental matrix $\mathbf{F}$ can be derived as follows [33]:

- Find a vector $\mathbf{C}$ such that $\mathbf{P}_1\mathbf{C} = 0$. The vector $\mathbf{C}$ is the null space of $\mathbf{P}_1$. It is also the homogeneous coordinate of the center of the first camera in the 3D coordinate.
- Calculate $\mathbf{e} = \mathbf{P}_2\mathbf{C}$.
- Calculate $\mathbf{P}_1^{\dagger}$, the pseudo-inverse of $\mathbf{P}_1$.
- For each three dimensional vector $\mathbf{e} = [e_1, e_2, e_3]^{\mathrm{T}}$, denote $[\mathbf{e}]_\times$ as:

$$[\mathbf{e}]_\times = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix}.$$

  For any vector $\mathbf{a}$ of length 3, such matrix satisfies $[\mathbf{e}]_\times\mathbf{a} = \mathbf{e} \times \mathbf{a}$, in which $\times$ denotes cross product. In other words, $[\mathbf{e}]_\times$ transforms a cross product to matrix multiplication.

- The fundamental matrix $\mathbf{F} = [\mathbf{e}]_\times\mathbf{P}_2\mathbf{P}_1^{\dagger}$.

In our experiments, if $\mathbf{x}_1$ and $\mathbf{x}_2$ form a correct corresponding pair, then the absolute value of $\mathbf{x}_2^{\mathrm{T}}\mathbf{F}\mathbf{x}_1$ is always less than 200. If a pair is wrongly formed, such value can be larger than 10,000. As a result, we set threshold to be 300, and if $|\mathbf{x}_2^{\mathrm{T}}\mathbf{F}\mathbf{x}_1| > 300$, $\mathbf{x}_1$ and $\mathbf{x}_2$ will be regarded as a wrongly identified pair.

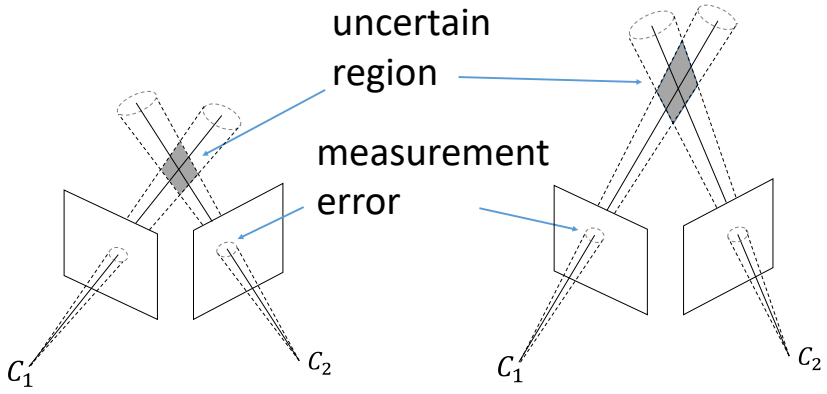


Figure 6.6: Uncertainty due to the measurement errors in stereo images.

Find vicinal LIDAR points: The last step of the corresponding point method is to check whether there exist LIDAR points within a cuboid with its center located at the calculated 3D position. The size of the cuboid is guided by the following observations: (1) The further the distance to the LIDAR origin, the sparser LIDAR points will be. Moreover, LIDAR points reflected by the ground are much sparser than points reflected by aboveground objects. Such phenomena are illustrated in the 3D LIDAR point cloud in Figure 6.3a. (2) Image coordinates errors in corresponding points are unavoidable. But the severity of measurement errors highly depends on the location of the 3D points, as illustrated in Figure 6.6. The shaded area illustrates the uncertainty region, which depends on the angle between the rays passing through camera centers. Points are less precisely localized if rays become more parallel. Besides, the scales of uncertainty in x, y, z directions are different. The x direction is the most sensitive, while the z direction is the least.

Denote the Euclidean distance between the calculated point and origin of the LIDAR coordinate system on the xy plane as d (unit: meter). In our experiments, we ignore the calculated points with d greater than 50 meters, for the reason that LIDAR points at those distances are too sparse and the measurement errors are too large for consistency checking. For d less than 50 meters, we choose the cuboid with length, width and height (parallel to x, y, z

Algorithm 2 Hierarchical Interpolation Algorithm

Input: 3D LIDAR point coordinates and calibration matrix.
Output: Distance image $\mathbf{A}$.
Separate LIDAR points as ground points and aboveground points.
Project ground points onto distance image $\mathbf{A}$, using a linear interpolation method to fill in missing values.
Initialize the remaining elements in $\mathbf{A}$ to ∞.
Run DBSCAN for aboveground points.
for each LIDAR cluster **do**
    (1) Project cluster points onto a new distance image $\mathbf{A}_{\text{temp}}$ of the same size using the calibration matrix.
    (2) Linear interpolate the projected LIDAR points.
    (3) Use morphological methods to preserve the object shape.
    (4) $\mathbf{A} = \min(\mathbf{A}, \mathbf{A}_{\text{temp}})$, min denotes element-wise minimum.
end for
return A.

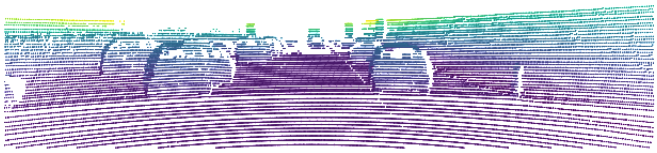
axis) equal to $0.1d$, $0.05d$ and $0.1 + 0.003d$, respectively. For each pair of corresponding points, if LIDAR points exist within this cuboid, we call that pair a *good corresponding pair*. Otherwise, we call it a *bad corresponding pair*.

6.6 Sensor Reliability Evaluation in LIFE

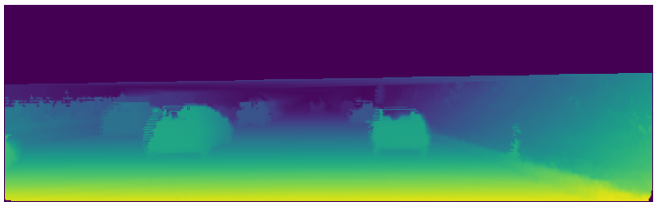
6.6.1 LIDAR Data Interpolation

Conventional interpolation methods such as linear interpolation, upsampling using kernel filters [27], interpolate a missing value point based on the values of all its neighbors. Hence, some interpolated values are calculated using values projected from different objects. Such phenomena degrade the interpolation performance significantly, as illustrated in Figure 6.7b.

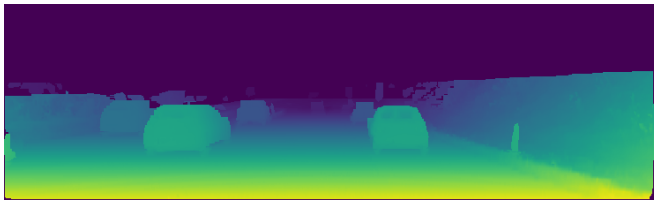
Algorithms proposed in [19, 75] can solve the interleaving issue by incorporating the corresponding camera images in the LIDAR interpolation process. However, these algorithms



(a) LIDAR points projected onto image plane.



(b) Standard linear interpolation results.



(c) Hierarchical interpolation results.

Figure 6.7: Performance comparison between conventional linear interpolation method and hierarchical interpolation method.

will not work when the cameras are under PEAs. To solve these issues, we propose a *hierarchical interpolation algorithm* without the involvement of image data. The overall steps are summarized in Algorithm 2, and the performance is shown in Figure 6.7c. Instead of interpolating projected LIDAR points altogether, the hierarchical interpolation algorithm interpolates projected LIDAR clusters obtained from DBSCAN independently and assembles them to construct the distance image. We first project all ground points onto a distance image and run the linear interpolation algorithm. Then for each LIDAR cluster, project all cluster points onto distance image and linearly interpolate them. Only the nearest distance value is kept if multiple (interpolated) points are projected onto the same pixel. This procedure continues until all clusters are assembled.

However, the 2D linear interpolation method will interpolate all objects as polygons. In

order to solve this deformation issue, we borrow the idea of morphological dilation and erosion [90]. First, change the projected LIDAR cluster points to a binary image, which 1 indicates a projected LIDAR point exists in that position, and 0 otherwise. Second, dilate the binary image with a filter of size 13×1, following by dilating the binary image with a filter of size 1×9. The size of the filter is chosen based on the gap between adjacent projected LIDAR points. Third, fill in regions of zeroes if they are enclosed by ones. This step is to reduce the prediction errors caused by holes inside objects. Finally, erode the binary image with the same filters and multiply it to the interpolation results elementwisely.

6.6.2 Data Prediction Using Deep Learning

Camera image and distance image prediction are both based on PredNet introduced in [56]. Distance maps are regarded as one-channel images to process. We train two separate neural networks for predicting camera and distance images. Both neural networks take data from 9 continuous time instances as input and output the predicted data for the next time instance.

The objective of PredNet is to predict the Nth image i_N based on the previous $N-1$ images $i_1, i_2, \cdots, i_{N-1}$. The architecture is illustrated in Figure 6.8. PredNet is composed of N stack modules, where each stack module is composed of L layers of (sub)modules. Each module of the network consists of four parts: an input convolutional layer A_l, a recurrent representation layer R_l, a prediction layer $\hat{A}_l$, and an error representation layer E_l. The lowest layer A_0^n is set to the image sequence itself, i.e., $A_0^n = i_n, \forall n \in \{1, 2, \cdots, N\}$. $\hat{A}_0^n$ is the prediction for i_n. PredNet is trained to minimize the weighted sum of L1 error between A_0^n and $\hat{A}_0^n$. The loss function is expressed as:

$$\text{Loss} = \frac{1}{N-1} \sum_{n=2}^{N} \|A_0^n - \hat{A}_0^n\|_1.$$

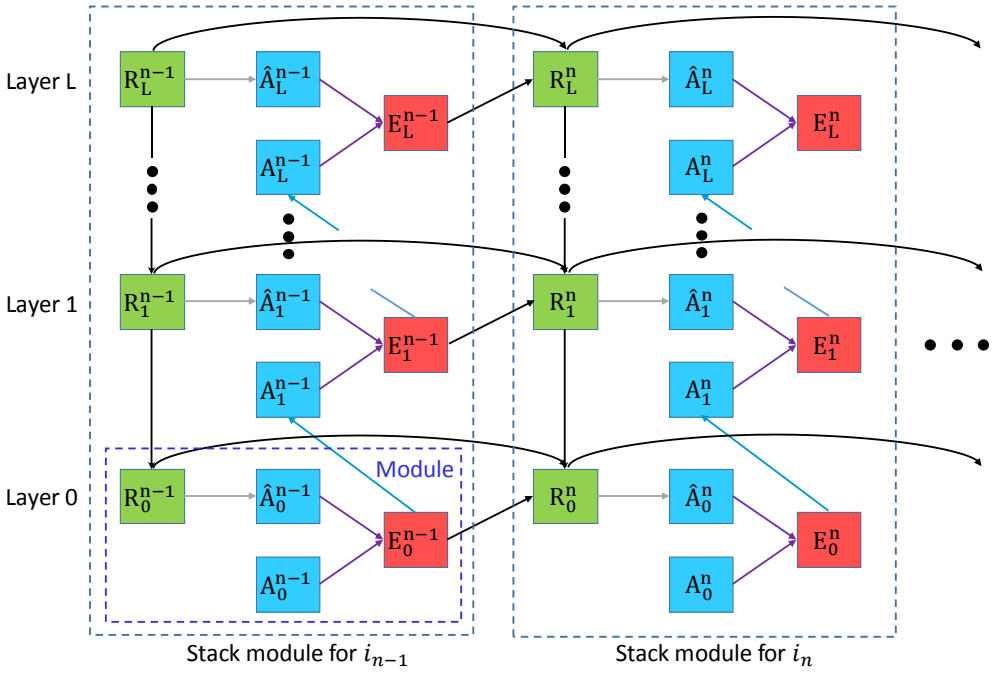


Figure 6.8: Architecture of PredNet.

where $\|\cdot\|_1$ denotes the L1 norm. The prediction error $\|A_0^1 - \hat{A}_0^1\|_1$ is not taken into account because without seeing any images, the prediction is a fixed value $\hat{A}_0^1 = 0$.

The parameter update procedure is described in Algorithm 3. The dimension of each component of PredNet is summarized in Table 6.1. In each layer l, the parameters are the same for all $n \in \{1, 2 \cdots, N\}$. The inputs have dimensionality of $(504, 152, 3)$ for camera images, and dimensionality of $(504, 152, 1)$ for distance images. In LIFE, we choose $N = 10$ and $L = 4$. In all convolutional layers, the size of the kernel is set to be 3×3 with stride equal to 1. The max pooling layer uses a stride of size 2×2. SatLU denotes saturating non-linearity unit ($\text{SatLU}(x, p_{\max}) = \min(x, p_{\max})$) and is utilized as the activation function for A_0^n with $p_{\max} = 1$ because pixel values cannot exceed 1. The number of channels in E_l is double the size of other components in the same layer because it is a concatenation of positive error term $\text{ReLU}(A_l^n - \hat{A}_l^n)$ and negative error term $\text{ReLU}(\hat{A}_l^n - A_l^n)$. Upsampling is performed

Algorithm 3 PredNet States Updates

Input: $A_0^n = i_n, \forall n,\ E_l^0 = 0, R_l^0 = 0$
Output: Prediction $\hat{A}_0^N$
for $n = 1$ to N **do**
    $R_L^n = \text{ConvLSTM}(E_L^{n-1}, R_L^{n-1})$
    for $l = L - 1$ to 0 **do**
        $R_l^n = \text{ConvLSTM}(E_l^{n-1}, R_l^{n-1}, \text{Upsample}(R_{l+1}^n))$
    end for
    for $l = 0$ to L **do**
        if $l = 0$ **then**
            $\hat{A}_0^n = \text{SatLU}(\text{ReLU}(\text{Conv}(R_0^n)))$
        else
            $\hat{A}_l^n = \text{ReLU}(\text{Conv}(R_l^n))$
        end if
        $E_l^n = [\text{ReLU}(A_l^n - \hat{A}_l^n); \text{ReLU}(\hat{A}_l^n - A_l^n)]$
        if $l < L$ **then**
            $A_{l+1}^n = \text{MaxPool}(\text{Conv}(E_t^n))$
        end if
    end for
end for

by interpolating missing values using the nearest neighbor method.

The recurrent representation layers R_l^n are updated according to R_l^{n-1}, E_l^{n-1}, as well as R_{l+1}^n. The updates are based on ConvLSTM [35], which is a recurrent neuron just like LSTM, but the internal matrix multiplications are substituted with convolution operations. The three components are concatenated and form a tensor to be used as the input of ConvLSTM neuron. In our implementations, the activation function of the cell state in ConvLSTM is tanh, and the activation functions for other states are hard sigmoid functions ($f(x) = 0$ if $x < -2.5$, $f(x) = 1$ if $x > 2.5$, $f(x) = 0.2x + 0.5$ if $-2.5 \leq x \leq 2.5$). Although PredNet outputs all prediction results for $n \geq 2$, only A_0^N is useful in LIFE.

Models are trained using data from more than 40,000 time instances in the KITTI dataset. Sequences of 10 time instances data are sampled. 55 recording sessions are used for training, 4 are used for validation, and 5 are used for testing. Two Titan XP GPU of 12 GB memory

Table 6.1: PredNet Implementation Details.

Layer	Components	RGB image dim	Distance image dim
$l = 0$	$A_0, \hat{A}_0, R_0$	$(504, 152, 3)$	$(504, 152, 1)$
	E_0	$(504, 152, 6)$	$(504, 152, 2)$
$l = 1$	$A_1, \hat{A}_1, R_1$	$(252, 76, 48)$	$(252, 76, 48)$
	E_1	$(252, 76, 96)$	$(252, 76, 96)$
$l = 2$	$A_2, \hat{A}_2, R_2$	$(126, 38, 96)$	$(126, 38, 96)$
	E_2	$(126, 38, 192)$	$(126, 38, 192)$
$l = 3$	$A_3, \hat{A}_3, R_3$	$(63, 19, 192)$	$(63, 19, 192)$
	E_3	$(63, 19, 384)$	$(63, 19, 384)$

are used for training. We resize all camera and distance images to 504×152 pixels to further reduce the computation resources and computation time. Adam optimization [41] using default parameter settings with batch size 4 is applied. Prediction performance is illustrated in Figure 6.3c and Figure 6.9.

6.6.3 Evaluation of Sensor Reliability

For each detected inconsistency, we compare the difference between the predicted data and the actual data to evaluate which sensory data is unreliable. For distance images, we use absolute value error to measure the differences. To mitigate the prediction errors caused by boundaries of objects, we apply a minimum filter of size 3×3 to the absolute error map. Values are scaled to $[0, 1]$, in which 0 indicates no difference, and 1 indicates the difference is greater than 50 meters. For camera images, we use structural similarity (SSIM) index [101] to measure the differences. SSIM is more representative than mean square error and absolute error in image processing. It is a perception-based metric that considers image degradation as a perceived change in structural information, while also incorporating perceptual phenomena such as luminance and contrast. SSIM is computed locally within a sliding window that moves pixel by pixel across the image, resulting in a *SSIM map*. The

value of SSIM index is bounded in $(0, 1]$, where 1 is reachable only when two images are identical in the sliding window, and 0 indicates no structural similarity. An absolute error map example and SSIM map example are illustrated in Figure 6.3c.

In the object matching method, we calculate the boundaries of the unmatched objects on the resized images. In SSIM map, if the average SSIM index in the bounded region is lower than 0.5, camera data is considered to be unreliable. In an absolute error map, if the number of pixel values greater than 0.1 exceeds 300 or two-thirds of the bounded region size, LIDAR data is considered to be unreliable. In the corresponding point method, for each bad corresponding pair, project the calculated 3D coordinate onto the image plane and calculate the absolute error in a 13×13 square centered at the projected point. If more than 100 pixels in the bounded region have values greater than 0.1, LIDAR is determined to be unreliable. The inconsistencies caused by object detection errors are ignored by LIFE since the sensory data still maintain the sequential correlation.

But the newly appeared object cannot be predicted at the first time instance it appears on the image (e.g., another car passing by on the adjacent lane). Such cases always impact the evaluation results on the boundary of images and are likely to make LIFE generate evaluation errors. To solve this issue, we ignore the prediction errors in the leftmost and rightmost 10 columns of SSIM maps and absolute error maps. Such modifications have no impact on PEA detection because it is infeasible to attack a sufficient amount of data points that can be projected within such small regions.

6.7 Performance Evaluation of LIFE

6.7.1 Emulation of Perception Error Attacks

We realize that the ideal way to evaluate the efficacy of LIFE is to use datasets obtained from experiments performed under real-world conditions. Unfortunately, we were not able to conduct such experiments due to our lack of access to an autonomous vehicle and testbed. We are aware that several recent studies have studied PEAs or similar security issues with experimental data [7, 80, 92, 108]. However, it should be noted that those experiments were conducted using sensor modules detached from a vehicle or using sensor modules attached to a stationary vehicle. We would not be able to evaluate the efficacy of LIFE with such data (collected from a detached sensor module or a stationary vehicle), because LIFE needs to be fed data collected by sensors on a moving vehicle that reflects the dynamism of the vehicle's surrounding environment as it travels through. For this reason, we resorted to using datasets that emulate PEAs, knowing full well the limitations of such an approach. In order to maximize the credibility of our emulated datasets, we emulated PEAs by carefully modifying the raw KITTI datasets based on the experimental results reported in the aforementioned studies. We would like to emphasize that the emulated PEAs were based on experimental results reported in prior studies, and were not based on our arbitrary assumptions.

1. *Emulating LIDAR spoofing and saturation attacks.* The spoofing attack is emulated based on experimental results introduced in [7], and the saturation attack is emulated based on [92]. As shown in Figure 6.1, the attack laser(s) inject pulses to LIDAR receiver apertures at several horizontal degrees to spoof or conceal points within this angle. As discussed in [7], to effectively fool the data processing algorithm, the spoofed points should be scattered around 8° horizontal angle in the front view. Based on these results, we first randomly select two rays in the forward direction that originated from the LIDAR coordinate system origin with

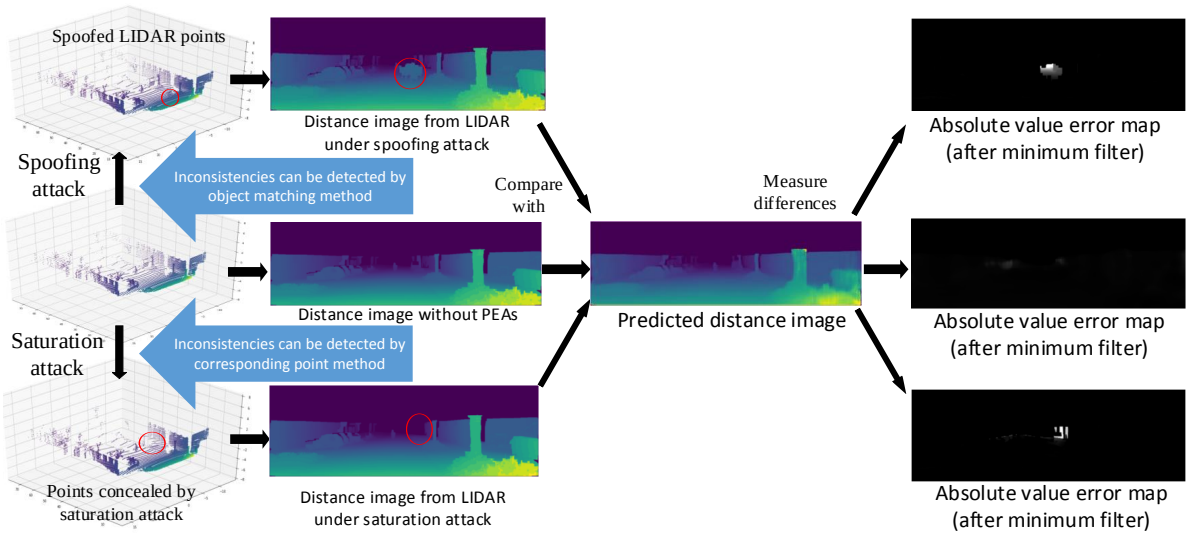
an angle of 8°. To emulate spoofing attacks, we randomly choose a distance between $5-15$ m and generate 80-120 faked points at that distance between the two selected rays. The heights are no more than 1.7 meters, the typical height of a vehicle. To emulate saturation attack, we remove all *aboveground points* between the two selected rays. Examples of point cloud under PEAs are illustrated in Figure 6.9.

2. *Emulating PEAs targeting cameras.* The PEAs targeting cameras are generated using image processing software. We emulated the effect of a strong light beam shooting towards a camera, as illustrated in Figure 6.9.

3. *Emulating PEAs caused by other attack methods and sensor malfunctions.* We also evaluated the performance of LIFE under two other PEAs that may potentially happen in practice: (1) Distance error attack, caused by moving a LIDAR apparatus to its calibrated position so that LIDAR points appear nearer or further than their actual positions. To emulate this PEA, we randomly choose two rays with angle 8° and move all LIDAR points between the two rays $10-15$ meters away. (2) Rotation error attack, caused by rotating the sensors so that LIDAR or camera slightly deviates from its calibrated position. To emulate this type of PEA, we rotate all LIDAR points 3.5° clockwise. Since LIFE allows some measurement errors when checking the consistency between LIDAR and camera, it has limited capability to detect rotation errors with an angle less than 3°.

4. *Description of Dataset.* We use the same datasets to evaluate LIFE performance as those used for testing image/LIDAR data prediction performance in section 6.6.3. They contain the following five driving scenarios:

- Dataset 1: Highway traffic with a lot of moving vehicles.
- Dataset 2: Open area with very few vehicles passing by.
- Dataset 3: Business street with many pedestrians.



(a) How LIFE detects PEAs targeting LIDAR.

(b) How LIFE detects PEAs targeting camera.

Figure 6.9: Illustration of how LIFE detects PEAs targeting LIDAR and camera.

- Dataset 4: School campus.

- Dataset 5: Residential area with only a few moving vehicles and many vehicles parked along the road.

These five datasets cover most of the real-world driving scenarios. Since data from consecutive time instances are of little difference, we observe very similar detection results for data from two or three consecutive time instances occasionally. Therefore, we further select data for testing LIFE performance every five time instances ($\approx$ 0.5s). We denote the collection

of LIDAR data and stereo image data at each time instance as a *bundle*. The generated PEAs are based on data from these selected bundles. The number of testing bundles in each dataset is summarized in Table 6.3.

We generate LIDAR spoofing, saturation, distance error, and rotation error attacks in all testing bundles. For camera PEAs, we only modified images in the first 20 bundles of each dataset. The reason is that the SSIM index is very sensitive to changes in luminance, which makes LIFE extremely effective in detecting camera PEAs. It is unnecessary to manually modify images using software for all testing bundles. Performance of LIFE under non-adversarial and adversarial scenarios is illustrated in Figure 6.9. More performance results under different scenarios are illustrated in Figure 6.10 and Figure 6.11. In example (a) and (c), which have a higher vehicle and pedestrian density, saturation attacks targeting LIDAR are emulated for demonstration. In example (b), (d), and (e), spoofing attacks targeting LIDAR are emulated for demonstration.

6.7.2 Limitations in Existing Sensor Fusion Algorithms

In this subsection, we present findings from our experiments to evaluate the performance of F-PointNet [84], MV3D [12], and AVOD [45] when PEAs are launched. The purpose of these algorithms is to estimate the 3D positions of cars, pedestrians, and cyclists.

Although average precision (AP) is the most widely used metric to evaluate object detection algorithms, it is not sufficient to demonstrate how the algorithms react to PEA instances. In our emulated LIDAR PEAs, the falsified LIDAR points may only impact the detection of one or even no existing objects. Nevertheless, there exist multiple objects in each testing bundle. Failing to detect one object correctly only decreases the AP slightly. Therefore, we evaluated the performance of prior sensor fusion algorithms by counting the number of

Table 6.2: Fraction of PEA instances that cause detection failures.

	F-PointNet	MV3D	AVOD
Camera blinding attack	0.99	0.98	0.99
LIDAR spoofing attack	0.27	0.23	0.26
LIDAR saturation attack	0.49	0.43	0.53
LIDAR distance error attack	0.46	0.48	0.50
LIDAR rotation error attack	0.79	0.83	0.84

testing bundles that contain one or more detection failures due to the existence of PEAs. For example, suppose a sensor fusion algorithm detects six out of seven objects correctly[1] in the non-adversarial scenario in a testing bundle. When LIDAR suffers a saturation attack, the algorithm is able to detect only four out of seven objects correctly. This indicates that a PEA was responsible for degrading the sensor fusion algorithm's performance. Such a metric can better describe how the PEAs impact the detection accuracy. We use the testing bundles used in section 6.7.1 to evaluate the performance of existing LIDAR and camera data fusion algorithms. For each type of PEA, we summarize the fraction of PEA instances that cause detection failures (i.e., the number of bundles that PEAs cause detection failures over the total number of testing bundles) in Table 6.2.

From Table 6.2, we observe that when cameras suffer blinding attacks, all the evaluated LIDAR and image data fusion algorithms fail to detect objects reliably. In other words, these sensor fusion algorithms are very vulnerable to camera PEAs. On the other hand, these algorithms are robust to LIDAR spoofing attacks. They can counter the impacts of LIDAR saturation attacks and distance error attacks to some extent as well. One possible reason for these results is that these algorithms focus on detecting cars, pedestrians, and cyclists. If the spoofed or eliminated points are not in the vicinity of existing LIDAR points associated with the target objects, they are unlikely to impact the detection results. Since

[1]An object is detected correctly if its category is classified correctly and the overlap between the 3D bounding box and the ground truth exceeds a threshold. The thresholds are different for cars, pedestrians, and cyclists in the KITTI benchmark

rotation error attacks modify the positions of all the aboveground LIDAR data points, prior algorithms have very limited capability in mitigating their impacts. In conclusion, existing algorithms are somewhat effective in mitigating some types of PEAs, but not all.

6.7.3 LIFE Performance in Non-adversarial Scenarios

Object detection algorithms can only detect certain types of objects listed in the training labels, such as cars, persons, etc. They do not detect other objects not included in the training labels, such as trees, walls, barriers, etc. Therefore, inconsistencies are always detected without the existence of PEAs. But LIFE is designed to tolerate such inconsistencies in the sensor reliability evaluation step. If LIFE detects that either LIDAR or camera data is anomalous under non-adversarial scenarios, this is an instance of a *false alarm*.

The performance of LIFE with and without PEAs at a specific time instance is illustrated in Figure 6.9. For camera images under non-adversarial settings, the average value in the whole SSIM map is always greater than 0.8. For distance images under non-adversarial settings, due to the imperfections in prediction algorithms, there exist some large values in the absolute value error map (shown as white dots). However, these large values are scattered and sparsely distributed. After applying the minimum filter, almost all of the errors are smoothed to values very close to 0.

We recorded the average number of overall corresponding pairs and bad corresponding pairs for all the tested bundles in each testing dataset. Results are summarized in Table 6.3. We can observe that even though sensors do not suffer PEAs, there still exist several bad corresponding pairs, especially in open areas such that objects are far away from the vehicle (the driving scenario in Dataset 2). The reason is that imperfect point coordinate measurements can cause errors when we calculate the 3D position from a pair of corresponding points. The

Table 6.3: Performance of LIFE in non-adversarial and adversarial scenarios.

	Metric description	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5
Dataset overview	Number of bundles for testing	167	78	212	46	508
	Avg number of total corresponding pairs	211.0	435.1	238.8	395.5	365.0
Non-adversarial scenarios	Avg number of bad corresponding pairs	5.09	6.12	0.93	3.09	3.03
	False alarms	2	1	2	0	3
Distance error attack	Avg number of bad corresponding pairs	14.77	12.83	16.32	16.54	11.85
	Correctly detect PEAs targeting LIDAR	167	73	212	45	500
Rotation error attack	Avg number of bad corresponding pairs	44.77	12.95	13.63	15.32	17.45
	Correctly detect PEAs targeting LIDAR	167	78	212	45	508
Saturation attack	Avg number of bad corresponding pairs	13.22	13.90	11.61	12.39	10.87
	Correctly detect PEAs targeting LIDAR	167	69	212	44	497
Spoofing attack	Correctly detect PEAs targeting LIDAR	158	76	191	44	496

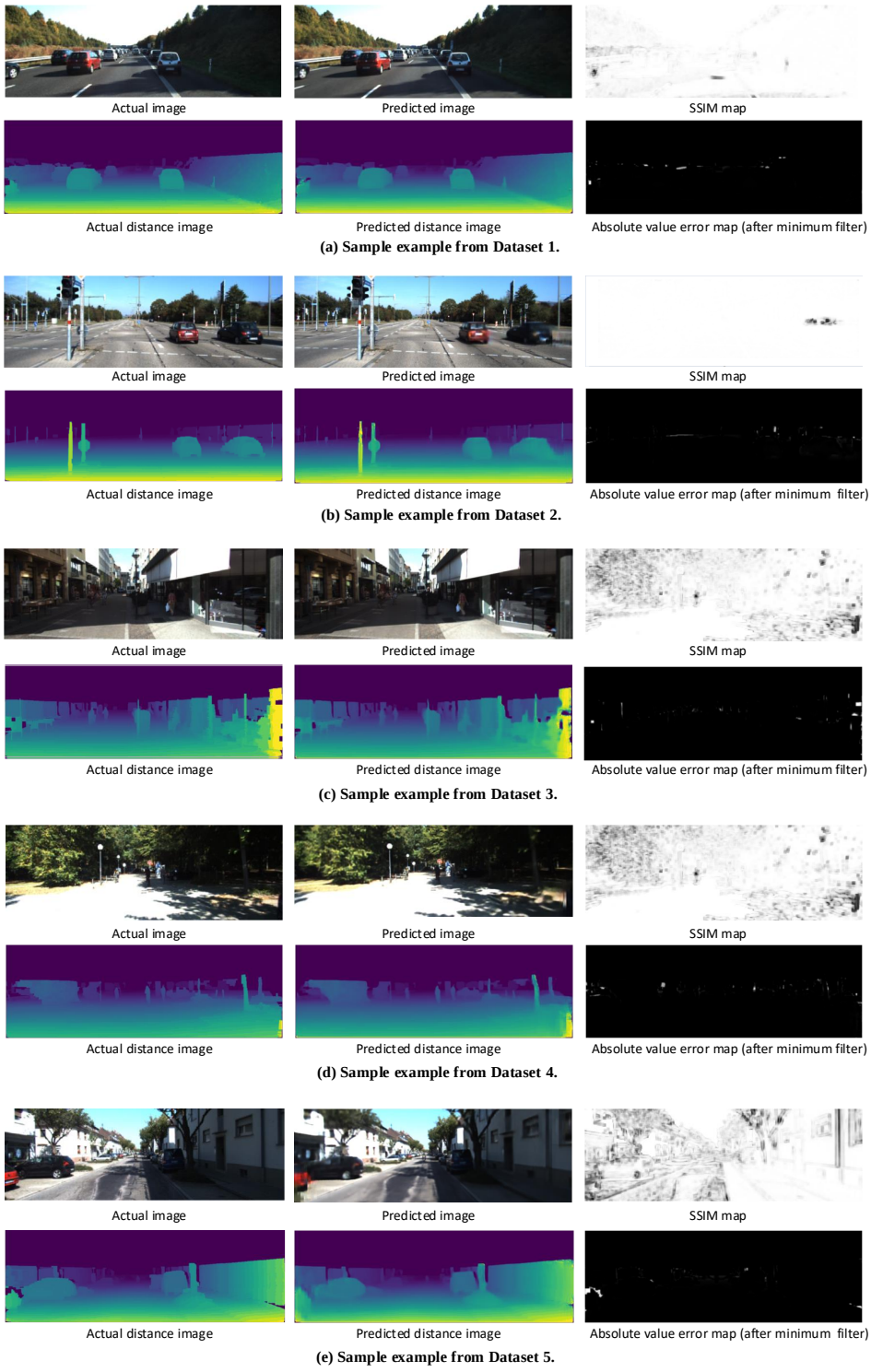
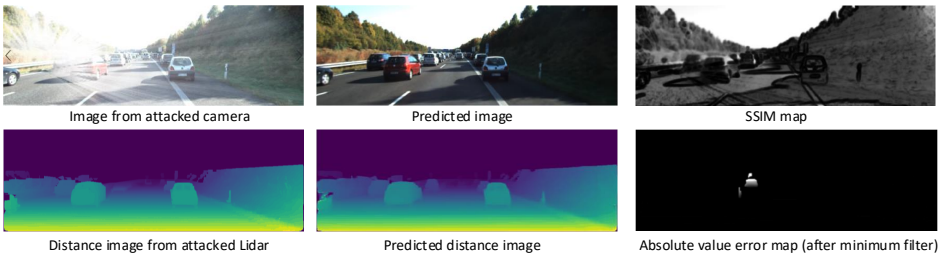
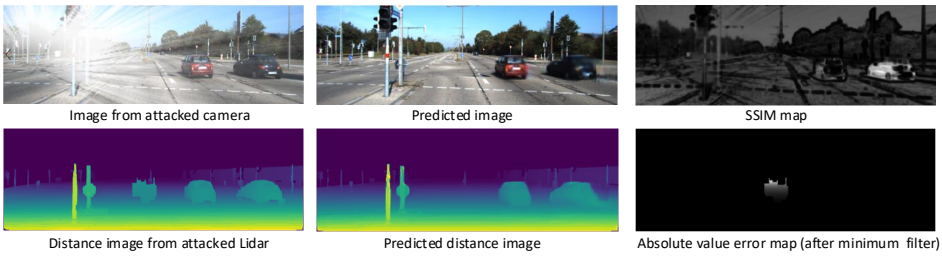


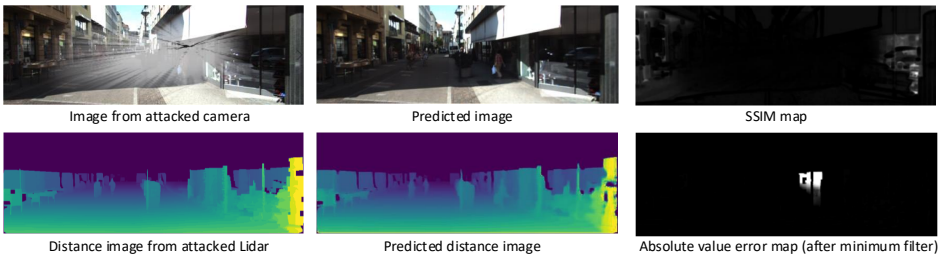
Figure 6.10: Performance of LIFE under non-adversarial scenarios.



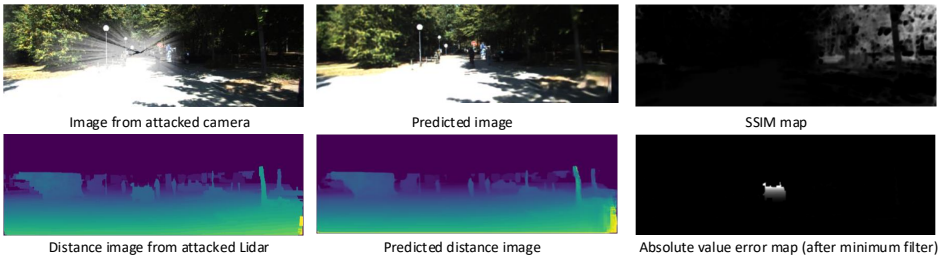
(a) Sample example from Dataset 1.



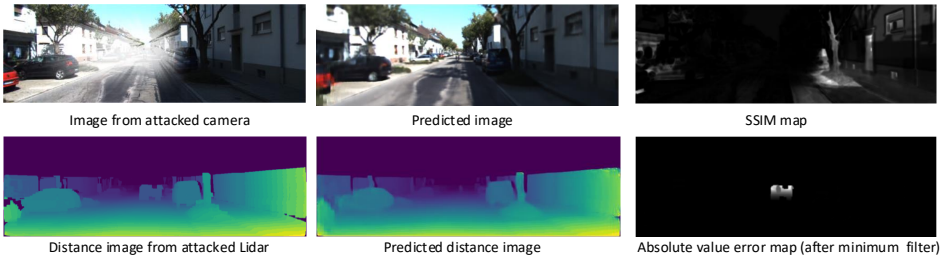
(b) Sample example from Dataset 2.



(c) Sample example from Dataset 3.



(d) Sample example from Dataset 4.



(e) Sample example from Dataset 5.

Figure 6.11: Performance of LIFE under adversarial scenarios.

further the calculated 3D position, the larger the error could be. More surrounding objects can help reduce the number of bad corresponding pairs, while fewer objects near the vehicle will increase the number.

As shown in Table 6.3, LIFE generates a small number of false alarms. We observe eight false alarms out of 1011 test bundles, and all of them identify LIDAR as unreliable. The camera is never falsely determined to be unreliable under non-adversarial scenarios, for the reason that the SSIM map is sufficiently distinguishable between adversarial and non-adversarial scenarios. False alarms are due to the following reasons: (1) The LIDAR prediction algorithm is inaccurate so that the bounding regions formed by unmatched LIDAR clusters or bad corresponding pairs happen to have high values in the absolute error map. (2) When the vehicle makes a turn, there may exist unmatched LIDAR clusters corresponding to newly appeared objects in the middle of images, which cannot be predicted correctly based on historical data. Due to the large error in absolute error maps, such instances are recognized as under PEAs in the sensor reliability evaluation step. However, all LIDAR points associated with false alarms are either at least 10 meters away to the left or right of the vehicle or at least 30 meters away in front of the vehicle. We assert that such false alarms barely impact the applicability of LIFE. LIFE does not generate false alarms in the vicinity of the vehicle, while attacks in nearer regions are expected to have larger safety impacts. Autonomous driving systems can focus more on the PEA detection results in the near region to alleviate the impacts of false alarms.

6.7.4 LIFE Performance in Adversarial Scenarios

Performance of LIFE against LIDAR PEAs: The performance of LIFE under adversarial scenarios is also illustrated in Figure 6.9. When LIDAR is under PEAs, we can

observe that the white pixels (large values) in an absolute value error map are clustered together. After applying the minimum filter, the cluster will remain there, which is very different from non-adversarial scenarios. The average number of bad corresponding pairs and the number of bundles that LIFE correctly detects LIDAR PEAs in each dataset are summarized in Table 6.3. The number of bad corresponding pairs is not listed for spoofing attack, because injecting fake echoes won't impact existing LIDAR points and the number of bad corresponding pairs will not be increased. In other PEA scenarios, the number of bad corresponding pairs is significantly increased. In other words, a much greater degree of inconsistencies is detected by LIFE when there exist PEAs targeting LIDAR.

However, LIFE may sometimes fail to detect PEAs targeting LIDAR. Our results indicate that all the failures can be categorized into the following three categories: (1) Eliminated or incorrectly positioned LIDAR points happen to project onto regions that no corresponding pairs are formed. Hence, no inconsistencies between LIDAR and camera will be detected, so that LIFE will not execute the sensor reliability evaluation step. However, such an occurrence is rare—we only observed three such cases out of 1011 testing bundles. (2) Most injected fake echoes/points are behind or very near existing aboveground objects. The induced fake objects can be detected by the object matching method. But when we construct a distance image, only the nearest distance value is kept if multiple (interpolated) points are projected onto the same pixel. Hence, spoofing attacks have limited impacts on the distance image. (3) The emulated LIDAR saturation attacks affect very few aboveground LIDAR points, hence no bad corresponding pairs related to such attacks are formed. In scenarios (2) and (3), the PEAs are expected to have almost no impact on the safety of a moving vehicle. As a consequence, it is not important to detect instances of such attacks. Based on the above discussions, we claim that instances of false dismissals are very unlikely to impact safety as those instances rarely affect the autonomous driving system's actions.

Performance of LIFE against camera PEAs: When cameras suffer PEAs, object detection algorithms recognize fewer objects, hence increase the number of inconsistent instances with LIDAR data. In the reliability evaluation step, since the SSIM index is very sensitive to the change in luminance, the values in SSIM map are always less than 0.2, which are much lower than those in non-adversarial scenarios (above 0.8). This phenomenon can be observed in Figure 6.9. In our experiments, LIFE was able to detect all of the 100 emulated PEAs targeting cameras.

Performance when both sensors are under PEAs: We consider a more capable attacker who can attack both types of sensors simultaneously. In this case, inconsistent instances can still be detected by the object matching method, because there always exist LIDAR clusters having no matched results obtained from object detectors. However, the corresponding point method doesn't work because LIFE can hardly find any corresponding pairs when cameras are under PEAs. To resolve this issue, if cameras are determined to suffer PEAs by the object matching method and the following reliability evaluation step, LIFE detects LIDAR PEAs by analyzing absolute error maps directly. This modified heuristic method works as follows. First, apply a 3×3 minimum filter to the error map. Then for each pixel, calculate whether there are more than 40 pixels having values greater than 0.1 in the surrounding 9×9 region with the selected pixel as the center. If the number of such pixels exceeds 400, LIFE determines that LIDAR is under PEAs. Although this modified method may result in an increase in false alarms, this more conservative approach is beneficial in terms of safety when the camera is under PEAs. We use the same 100 bundles as in section 6.3 to evaluate LIFE's performance when both sensors are under PEAs. All camera PEAs can be effectively detected. The LIDAR PEA detection ratio for spoofing, saturation, distance error, and rotation error attacks are 97%, 96%, 97%, 100%, respectively.

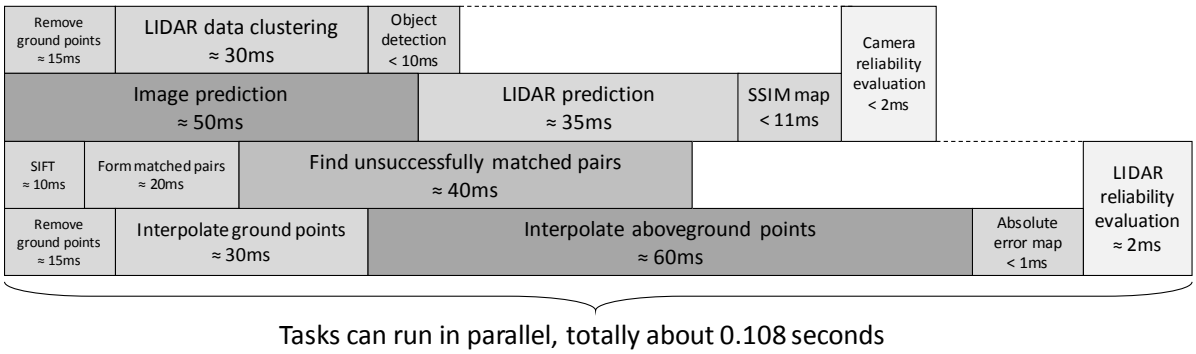


Figure 6.12: Task scheduling in LIFE.

6.7.5 Computation Time Analysis

We evaluated the performance of LIFE using one Intel Core i7 CPU and two Titan XP GPUs. We assert that this only represents a fraction of the computing power of real-world systems used for autonomous vehicles. For instance, the Nvidia autonomous driving platform uses two Volta GPUs and two Turing GPUs for robot taxi applications [69]. The overall computation power of these four GPUs is much higher than those used in our study. Moreover, Tesla is developing new AI chips, which are expected to be 7× faster than the Nvidia system in terms of TOPS (tera operations per second) [96]. Considering the computing power of current and emerging autonomous driving systems, we claim that the computation overhead of LIFE is reasonable, and does not pose an impediment to its adoption.

Most of the tasks in LIFE can be executed in parallel. The task scheduling and the average running time of each task are shown in Figure 6.12. The total execution time of one execution instance of LIFE is approximately 0.108 sec. Tasks such as detecting feature points, constructing the SSIM map, and data prediction can all be executed with high parallelization and efficiently executed by a GPU. Our software implementation of LIFE was written in Python using freely available packages, and the time to execute all tasks is roughly the same as the KITTI data sampling period. The bottleneck lies in the LIDAR interpolation

algorithm since packages using GPU to accelerate 2D interpolation have not been ready for public use yet. In commercial systems, it is common practice to employ optimization techniques, in terms of both software and hardware, to reduce the execution time [17].

6.7.6 Application of LIFE in Autonomous Driving Systems

In this subsection, we discuss how to use the LIFE results to enhance safety and security in real-world autonomous driving systems. Note that besides reporting whether LIDAR and camera are under PEAs, LIFE also outputs the detailed locations of the PEA instances, such as positions of the spoofed LIDAR points and the problematic regions of camera images. The remaining sensory data are still considered as reliable. In other words, LIFE specifies the untrustworthy data points rather than simply claiming whether sensors are under PEAs. The locations of the PEA instances can be served as additional inputs to the autonomous driving systems so that the driving systems can reevaluate the driving plan based on more trustworthy sensory data. For example, when LIFE determines that a cluster of LIDAR points are spoofed points, the driving system can ignore these points in the LIDAR data processing steps. Different approaches are required to handle LIDAR saturation attacks and camera blinding attacks since part of the sensory data are missing. The driving system either needs to predict the location of surrounding objects based on sensory data in previous timestamps or purely relies on normal sensors to make the safest driving decisions.

6.8 Chapter Summary

In this chapter, we propose LIFE to enhance the security and safety of autonomous vehicles by detecting PEAs targeting LIDAR and camera. The PEA detection is performed by

analyzing the correlation between 3D LIDAR data and 2D camera data. No additional sensors are required to incorporate LIFE into existing autonomous vehicles that are equipped with LIDAR and stereo cameras.

However, autonomous driving system is an extremely complicated system, which requires thousands of real-world scenarios to be taken into consideration. Here, we discuss some limitations of LIFE.

(1) Our emulated attacks are based on existing experimental results, which may deviate from actual scenarios. Moreover, we do not consider all plausible real-world scenarios. The KITTI dataset represents only driving conditions with good weather and an excellent field of view for the driver. Due to the lack of data, we were not able to evaluate the performance of LIFE in adverse driving conditions.

(2) Since several algorithms used in LIFE are not essential in navigating autonomous vehicles (e.g., predict camera and distance images), we need extra efforts to build the associated models to incorporate LIFE into existing autonomous driving platforms. Besides, the current execution time has not been fully optimized to meet the real-time requirements.

(3) The data prediction algorithms are not perfect and there exist a small number of false alarm instances that may impact real-world driving potentially. Since the sensor reliability evaluation results are highly dependent on the predicted data, a more reliable prediction algorithm is expected to further reduce the false alarm rates.

(4) An adversary may be able to circumvent LIFE if it launches PEAs that take effect gradually over a time period, during which time the impact of each attack instance is too insignificant to be detected by LIFE. The accumulated effect of the attack instances can cause the targeted autonomous driving system to malfunction. However, as noted in Section 6.3, such attacks are very difficult to carry out using existing technologies. In our threat model,

we assume that the adversary is not able to carry out such attacks.

In the future, we will incorporate LIFE into production-level autonomous driving platforms and evaluate the performance of LIFE in end-to-end evaluations. Such end-to-end evaluations help us better understand the computation overhead of LIFE and how much gap the current results towards practical situations. We will also analyze how to utilize intermediate sensor data processing results from the autonomous driving platforms to speed up LIFE.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Deep learning has demonstrated great potential in processing signals. The application domains range from image processing, sensor data processing, text mining, to newly emerged research areas such as wireless signal processing. Compare to traditional feature engineering techniques that require in-depth domain knowledge, deep learning is able to search for features that are helpful in learning tasks automatically by exploring the training data. Besides, deep learning can uncover new features and more complex features that humans can miss.

However, the security and reliability of deep learning models are impacted by many factors. In the training stage, the reliability of a deep learning model is largely determined by the size of training data. In the testing stage, deep learning algorithms are vulnerable to be fooled under adversarial scenarios. On the one hand, deep learning models are designed to only work under constrained circumstances. The results can be highly unreliable if a deep learning model is applied to improper application scenarios or adversarial scenarios. On the other hand, most deep learning models are built to work well under non-adversarial settings. which lacks the ability to check whether the input data are falsified by attackers.

In this dissertation, we have studied the aforementioned issues in three aspects. First, we study the relationship between model reliability and training sample size in deep learning-

based classification algorithms in wireless communications applications. Second, we discuss how OOD testing data impact the model reliability and propose a novel deep learning model called FOOD to detect OOD data. Finally, we analyze the impact of PEAs on deep learning-based signal processing targeting autonomous driving and propose a countermeasure called LIFE to enhance the security of the autonomous driving system.

7.2 Future Work

With the development of artificial intelligence, more and more security and reliability issues were discovered. In sections 4.8, 5.8 and 6.8, we summarize the limitations of our research works and propose several future research directions to improve our work. Besides improving our works, we find the following research problems to be very promising:

(1) A wide range of data has very strong sequential correlations, such as time series data and natural language data. It is common to apply RNN to process these data. Therefore, it will be interesting to analyze the relationship between recall for each class and the training sample size for each class in RNN-based deep learning models.

(2) FOOD only focuses on input data with high spatial correlations. It is insensitive to the order of inputs and cannot work with variable input sizes. As a consequence, a novel model architecture is required to detect OOD data in time-dependent deep learning applications, such as machine translation, speech recognition, time series analysis, etc.

(3) In autonomous driving systems, attackers can also fool the sensor data processing algorithms to cause accidents. For example, Evtimov *et al.* demonstrated that by sticking some white and black papers, a stop sign was misclassified as a speed limit 45 sign by road sign recognition systems [24]. Prior works have shown that images can be misclassified by

adding small perturbations that are unnoticeable to human beings [9, 30]. Such phenomena cause severe security risks in applying deep learning-based signal processing algorithms to safety-critical and security-critical real-world applications. There is an urgent need for an effective solution to fight against such attacks.

Bibliography

[1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

[2] Alireza Asvadi, Luis Garrote, Cristiano Premebida, Paulo Peixoto, and Urbano J Nunes. DepthCN: Vehicle detection using 3d-LIDAR and ConvNet. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–6. IEEE, 2017.

[3] Baidu. Baidu apollo. http://apollo.auto/, 2017.

[4] Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

[5] Natthaphan Boonyanunta and Panlop Zeephongsekul. Predicting the relationship between the size of training sample and the predictive power of classifiers. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 529–535. Springer, 2004.

[6] Ali Borji and Laurent Itti. Human vs. computer in scene and object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 113–120, 2014.

[7] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z Morley Mao. Adversarial sensor attack on lidar-

based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2267–2281, 2019.

[8] Yulong Cao, Chaowei Xiao, Dawei Yang, Jing Fang, Ruigang Yang, Mingyan Liu, and Bo Li. Adversarial objects against lidar-based autonomous driving systems. *arXiv preprint arXiv:1907.05418*, 2019.

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[10] Ricardo Omar Chavez-Garcia and Olivier Aycard. Multiple sensor fusion and classification for moving object detection and tracking. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):525–534, 2016.

[11] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[12] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, page 3, 2017.

[13] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1836–1843. IEEE, 2014.

[14] Junghwan Cho, Kyewook Lee, Ellie Shin, Garry Choy, and Synho Do. How much

data is needed to train a medical image deep learning system to achieve necessary high accuracy? *arXiv preprint arXiv:1511.06348*, 2015.

[15] Hyunsun Choi, Eric Jang, and Alexander A Alemi. Waic, but why? generative ensembles for robust anomaly detection. *arXiv preprint arXiv:1810.01392*, 2018.

[16] On-Road Automated Driving (ORAD) committee. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE International*, Jan 17, 2014.

[17] Xuewen Cui and Wu-chun Feng. Iterative machine learning (iterml) for effective parameter pruning and tuning in accelerators. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 16–23, 2019.

[18] Erik Daxberger and José Miguel Hernández-Lobato. Bayesian variational autoencoders for unsupervised out-of-distribution detection. *arXiv preprint arXiv:1912.05651*, 2019.

[19] Jennifer Dolson, Jongmin Baek, Christian Plagemann, and Sebastian Thrun. Upsampling range data in dynamic environments. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1141–1148. IEEE, 2010.

[20] Simon S Du, Yining Wang, Xiyu Zhai, Sivaraman Balakrishnan, Ruslan R Salakhutdinov, and Aarti Singh. How many samples are needed to estimate a convolutional neural network? In *Advances in Neural Information Processing Systems*, pages 373–383, 2018.

[21] Aveek Dutta and Mung Chiang. "See something, say something" crowdsourced enforcement of spectrum policies. *IEEE Transactions on Wireless Communications*, 15 (1):67–80, 2015.

[22] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data, 2017.

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[24] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

[25] Hamza Fawzi, Paulo Tabuada, and Suhas Diggavi. Secure estimation and control for cyber-physical systems under adversarial attacks. *IEEE Transactions on Automatic control*, 59(6):1454–1467, 2014.

[26] Rosa L Figueroa, Qing Zeng-Treitler, Sasikiran Kandula, and Long H Ngo. Predicting sample size required for classification performance. *BMC medical informatics and decision making*, 12(1):8, 2012.

[27] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4231, 2018.

[28] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11): 1231–1237, 2013.

[29] Robert Geirhos, Carlos RM Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge,

and Felix A Wichmann. Generalisation in humans and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 7538–7550, 2018.

[30] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015. URL http://arxiv.org/abs/1412.6572.

[31] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[32] The Guardian. Tesla driver killed while using autopilot was watching Harry Potter, witness says. https://www.theguardian.com/technology/2016/jul/01/tesla-driver-killed-autopilot-self-driving-car-harry-potter, July 1, 2016.

[33] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[34] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[35] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. In *International Conference on Learning Representations*, 2018.

[36] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[37] Soonmin Hwang, Namil Kim, Yukyung Choi, Seokju Lee, and In So Kweon. Fast multiple objects detection and tracking fusing color camera and 3d LIDAR for intel-

ligent vehicles. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2016 13th International Conference on*, pages 234–239. IEEE, 2016.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[39] Radoslav Ivanov, Miroslav Pajic, and Insup Lee. Attack-resilient sensor fusion for safety-critical cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):21, 2016.

[40] Krishna Karra, Scott Kuzdeba, and Josh Petersen. Modulation recognition using hierarchical deep neural networks. In *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–3. IEEE, 2017.

[41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[42] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[43] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[45] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1–8. IEEE, 2018.

[46] Merima Kulin, Tarik Kazaz, Ingrid Moerman, and Eli De Poorter. End-to-end learning from spectrum data: A deep learning approach for wireless signal identification in spectrum monitoring applications. *IEEE Access*, 6:18484–18501, 2018.

[47] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.

[48] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[49] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*, 2018.

[50] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177, 2018.

[51] Woongsup Lee, Minhoe Kim, and Dong-Ho Cho. Deep sensing: Cooperative spectrum sensing based on convolutional neural networks. *arXiv preprint arXiv:1705.08164*, 2017.

[52] Qingquan Li, Long Chen, Ming Li, Shih-Lung Shaw, and Andreas Nuchter. A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation

in challenging road scenarios. *IEEE Transactions on Vehicular Technology*, 63(2): 540–555, 2014.

[53] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018.

[54] Jinshan Liu and Jerry Park. "Seeing is not always believing": Detecting perception error attacks against autonomous vehicles. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[55] Jinshan Liu, Gaurang Naik, and Jung-Min Jerry Park. Coexistence of DSRC and Wi-Fi: Impact on the performance of vehicular safety applications. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[56] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[57] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[58] D Mader, R Blaskow, P Westfeld, and C Weller. Potential of UAV-based laser scanner and multispectral camera data in building inspection. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 41, 2016.

[59] Matlab. CWT-based time-frequency analysis. https://www.mathworks.com/help/wavelet/examples/cwt-based-time-frequency-analysis.html, 2019.

[60] Enrico Mattei, Cass Dalton, Andrew Draganov, Brent Marin, Michael Tinston, Greg Harrison, Bob Smarrelli, and Marc Harlacher. Feature learning for enhanced security

in the internet of things. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[61] Bryan Menegus and Kate Conger. Uber self-driving car struck and killed Arizona woman while in autonomous mode. https://gizmodo.com/uber-self-driving-car-killed-arizona-woman-while-in-au-1823891032, March 19, 2018.

[62] Kevin Merchant, Shauna Revay, George Stantchev, and Bryan Nousain. Deep learning for RF device fingerprinting in cognitive communication networks. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):160–167, 2018.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[64] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[65] Cyrille Morin, Leonardo S Cardoso, Jakob Hoydis, Jean-Marie Gorce, and Thibaud Vial. Transmitter classification with supervised deep learning. In *International Conference on Cognitive Radio Oriented Wireless Networks*, pages 73–86. Springer, 2019.

[66] Gaurang Naik, Jinshan Liu, and Jung-Min Jerry Park. Coexistence of dedicated short range communications (DSRC) and Wi-Fi: Implications to Wi-Fi performance. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[67] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2018.

[68] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[69] Nvidia. Nvidia Drive AGX developer kit. https://www.nvidia.com/en-us/self-driving-cars/drive-platform/, 2018.

[70] Taiwo Oyedare and Jung-Min Jerry Park. Estimating the required training dataset size for transmitter classification using deep learning. In *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–10. IEEE, 2019.

[71] Timothy J O'Shea, Johnathan Corgan, and T Charles Clancy. Convolutional radio modulation recognition networks. In *International conference on engineering applications of neural networks*, pages 213–226. Springer, 2016.

[72] Timothy James O'Shea, Tamoghna Roy, and T Charles Clancy. RF datasets for machine learning. https://www.deepsig.ai/datasets, 2018.

[73] Timothy James O'Shea, Tamoghna Roy, and T Charles Clancy. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):168–179, 2018.

[74] Gaurav Pandey, James R McBride, and Ryan M Eustice. Ford campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552, 2011.

[75] Kihong Park, Seungryong Kim, and Kwanghoon Sohn. High-precision depth estimation with the 3d lidar and stereo fusion. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2156–2163. IEEE, 2018.

[76] Simon Parkinson, Paul Ward, Kyle Wilson, and Jonathan Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):2898–2915, 2017.

[77] Kishor Patil, Knud Skouby, Ashok Chandra, and Ramjee Prasad. Spectrum occupancy statistics in the context of cognitive radio. In *2011 The 14th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–5. IEEE, 2011.

[78] Shengliang Peng, Hanyu Jiang, Huaxia Wang, Hathal Alwageed, Yu Zhou, Marjan Mazrouei Sebdani, and Yu-Dong Yao. Modulation classification based on signal constellation diagrams and deep learning. *IEEE transactions on neural networks and learning systems*, 30(3):718–727, 2018.

[79] Jonathan Petit and Steven E Shladover. Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):546–556, 2015.

[80] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. Remote attacks on automated vehicles sensors: Experiments on camera and LiDAR. *Black Hat Europe*, 11:2015, 2015.

[81] Adam C Polak, Sepideh Dolatshahi, and Dennis L Goeckel. Identifying wireless users via transmitter imperfections. *IEEE Journal on selected areas in communications*, 29 (7):1469–1479, 2011.

[82] Robi Polikar et al. The wavelet tutorial, 1996.

[83] Cristiano Premebida and Urbano Nunes. Fusing LIDAR, camera and semantic information: A context-based approach for pedestrian detection. *The International Journal of Robotics Research*, 32(3):371–384, 2013.

[84] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[85] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[86] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, pages 14680–14691, 2019.

[87] Shamnaz Riyaz, Kunal Sankhe, Stratis Ioannidis, and Kaushik Chowdhury. Deep learning convolutional neural networks for radio identification. *IEEE Communications Magazine*, 56(9):146–152, 2018.

[88] Giedre Sabaliauskaite and Jin Cui. Integrating autonomous vehicle safety and security. In *Proceedings of the 2nd International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2017), Barcelona, Spain*, pages 12–16, 2017.

[89] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[90] Jean Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.

[91] Yi Shi, Kemal Davaslioglu, Yalin E Sagduyu, William C Headley, Michael Fowler, and Gilbert Green. Deep learning for RF signal classification in unknown and dynamic spectrum environments. In *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–10. IEEE, 2019.

[92] Hocheol Shin, Dohyun Kim, Yujin Kwon, and Yongdae Kim. Illusion and dazzle: Adversarial optical channel exploits against Lidars for automotive applications. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 445–467. Springer, 2017.

[93] Yasser Shoukry, Pierluigi Nuzzo, Alberto Puggelli, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, and Paulo Tabuada. Secure state estimation for cyber-physical systems under sensor attacks: A satisfiability modulo theory approach. *IEEE Transactions on Automatic Control*, 62(10):4917–4932, 2017.

[94] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[95] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[96] Tesla. Tesla hardware 3 (full self-driving computer) detailed. https://www.autopilotreview.com/tesla-custom-ai-chips-hardware-3/, 2019.

[97] L Theis, A van den Oord, and M Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR 2016)*, pages 1–10, 2016.

[98] Vladimir Vapnik, Esther Levin, and Yann Le Cun. Measuring the VC-dimension of a learning machine. *Neural computation*, 6(5):851–876, 1994.

[99] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 550–564, 2018.

[100] Yu Wang, Miao Liu, Jie Yang, and Guan Gui. Data-driven deep learning for automatic modulation recognition in cognitive radios. *IEEE Transactions on Vehicular Technology*, 68(4):4074–4077, 2019.

[101] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[102] Zidong Wang, Dong Wang, Bo Shen, and Fuad E Alsaadi. Centralized security-guaranteed filtering in multirate-sensor fusion under deception attacks. *Journal of the Franklin Institute*, 355(1):406–420, 2018.

[103] Zining Wang, Wei Zhan, and Masayoshi Tomizuka. Fusing bird's eye view LIDAR point cloud and front view camera image for 3d object detection. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6. IEEE, 2018.

[104] Waymo. Google self-driving car project. https://waymo.com/, 2019.

[105] Wenwu Xie, Sheng Hu, Chao Yu, Peng Zhu, Xin Peng, and Jingcheng Ouyang. Deep learning in digital modulation recognition using high order cumulants. *IEEE Access*, 7:63760–63766, 2019.

[106] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196, 2018.

[107] Wenyuan Xu, Chen Yan, Weibin Jia, Xiaoyu Ji, and Jianhao Liu. Analyzing and enhancing the security of ultrasonic sensors for autonomous vehicles. *IEEE Internet of Things Journal*, 5(6):5015–5029, 2018.

[108] Chen Yan, Wenyuan Xu, and Jianhao Liu. Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle. *DEF CON*, 24, 2016.

[109] Khalid Youssef, Louis-S Bouchard, KZ Haigh, H Krovi, J Silovsky, and CP Vander Valk. Machine learning approach to RF transmitter identification. *arXiv preprint arXiv:1711.01559*, 2017.

[110] Yufeng Zhang, Wanwei Liu, Zhenbang Chen, Ji Wang, Zhiming Liu, Kenli Li, Hongmei Wei, and Zuoning Chen. Out-of-distribution detection with distance guarantee in deep generative models. *arXiv preprint arXiv:2002.03328*, 2020.

[111] David Zimmerer, Simon AA Kohl, Jens Petersen, Fabian Isensee, and Klaus H Maier-Hein. Context-encoding variational autoencoder for unsupervised anomaly detection. *arXiv preprint arXiv:1812.05941*, 2018.