AN ABSTRACT OF THE THESIS OF

Jared Gaskin for the degree of Master of Science in Electrical and Computer Engineering presented on June 16, 2022.

Title: Towards Deep Learning Model Portability for Domain-Agnostic Device Fingerprinting

Abstract approved: ________________________________

Bechir Hamdaoui     Lizhong Chen

In recent years, RF (Radio Frequency) device fingerprinting using deep learning has emerged as a method of identifying devices solely by their RF transmissions. Conventional approaches to this type of device fingerprinting are not portable to different domains. That is, if a model for this purpose is trained on data from one domain, the model will not perform well on data from a different but related domain. For RF fingerprinting, changing the receiver used, the day on which data was captured, or the configuration settings of transmitters all amount to changing the domain. This work proposes a technique that, using metric learning and a calibration process, enables a model trained with data from one domain to perform well on data from another domain. This is accomplished with only a small amount of training data from the target domain and without changing the weights of the model, which makes the technique computationally quick. This work evaluates the
effectiveness of the proposed technique on RF data captured using a testbed of real devices in a variety of different scenarios. The results of this evaluation show that this technique is viable and especially useful for applications where computational resources are restricted.
Towards Deep Learning Model Portability for Domain-Agnostic
Device Fingerprinting

by

Jared Gaskin

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 16, 2022
Commencement June 2023

APPROVED:

________________________________________
Co-Major Professor, representing Electrical and Computer Engineering

________________________________________
Co-Major Professor, representing Electrical and Computer Engineering

________________________________________
Director of the School of Electrical Engineering and Computer Science

________________________________________
Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

________________________________________
Jared Gaskin, Author
ACKNOWLEDGEMENTS

The author expresses sincere appreciation for the time and effort put forth in support of this work by both of the advising professors, Dr. Bechir Hamdaoui and Dr. Lizhong Chen, as well as Dr. Weng-Keen Wong.
# TABLE OF CONTENTS

1 Introduction 1

2 Motivating Use Cases 7
   2.1 Enabling Portability Across Different Receivers . . . . . . . . . . . . 7
   2.2 Enabling Portability Across Different Channels . . . . . . . . . . . . 9
   2.3 Enabling Portability Across Multiple Configurations . . . . . . . . . . . 9

3 Related Work 11
   3.1 Works Addressing Channel Portability . . . . . . . . . . . . . . . . 11
   3.2 Works Addressing Hardware Portability . . . . . . . . . . . . . . . 12
   3.3 Works Addressing Multiple Types of Portability . . . . . . . . . . . . 13

4 Domain-Agnostic Device Fingerprinting 15
   4.1 Deep Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
      4.1.1 Metric Learning and Twin Neural Networks . . . . . . . . . . 17
      4.1.2 Training the Twin Neural Network . . . . . . . . . . . . . . . 20
   4.2 Calibration and Decision Making Algorithms . . . . . . . . . . . . . . . 21
      4.2.1 Calibration Task . . . . . . . . . . . . . . . . . . . . . . . . 21
      4.2.2 Closed-set Decision Making Task . . . . . . . . . . . . . . . 23
      4.2.3 Open-set Decision Making Task . . . . . . . . . . . . . . . . 26

5 Experimental Scenarios and Dataset Collection 28
   5.1 Testbed . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
   5.2 Experimental Scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . 30

6 Experimental Results 34
   6.1 Neural Network Implementation . . . . . . . . . . . . . . . . . . . . . . 34
   6.2 Performance Evaluation Metrics . . . . . . . . . . . . . . . . . . . . . . 36
      6.2.1 Closed-set metrics . . . . . . . . . . . . . . . . . . . . . . . . 36
      6.2.2 Open-set metrics . . . . . . . . . . . . . . . . . . . . . . . . 36
   6.3 Closed-set Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
      6.3.1 Hardware Portability . . . . . . . . . . . . . . . . . . . . . . . 40
### TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.2 Channel Portability</td>
<td>44</td>
</tr>
<tr>
<td>6.3.3 Configuration Portability</td>
<td>45</td>
</tr>
<tr>
<td>6.4 Open-set Evaluation</td>
<td>50</td>
</tr>
<tr>
<td>6.4.1 Hardware Portability</td>
<td>51</td>
</tr>
<tr>
<td>6.4.2 Channel Portability</td>
<td>55</td>
</tr>
<tr>
<td>6.4.3 Configuration Portability</td>
<td>57</td>
</tr>
<tr>
<td>6.5 Effect of Number of Inputs per Decision</td>
<td>63</td>
</tr>
<tr>
<td>6.6 Effect of Amount of Calibration Data</td>
<td>65</td>
</tr>
<tr>
<td>6.7 Computational Speed Considerations</td>
<td>68</td>
</tr>
<tr>
<td>7 Open Research Challenges &amp; Directions</td>
<td>71</td>
</tr>
<tr>
<td>8 Conclusion</td>
<td>73</td>
</tr>
<tr>
<td>Bibliography</td>
<td>73</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Accuracy of DNN classifier models trained and tested on data from 10 transmitters under varying conditions.</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Transferring of a trained model to a different domain (hardware, channel, configuration) via calibration.</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>A high-level diagram of the proposed technique</td>
<td>15</td>
</tr>
<tr>
<td>4.2</td>
<td>A general twin neural network architecture using the triplet loss function during training.</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>Example of triplet loss values for one ‘anchor’ (A), ‘positive’ (P), and ‘negative’ (N) triplet</td>
<td>19</td>
</tr>
<tr>
<td>4.4</td>
<td>Depiction in 2-D of the calibration process for a single device, beginning after N examples from Device i have passed through the trained twin network.</td>
<td>21</td>
</tr>
<tr>
<td>4.5</td>
<td>Depiction in 2-D of calculation of A_i and B_i for one of the calibrated devices during the closed-set decision making algorithm.</td>
<td>23</td>
</tr>
<tr>
<td>5.1</td>
<td>(a) One of the two identical halves of the twin neural network architecture. (b) and (c) images of testbed hardware.</td>
<td>29</td>
</tr>
<tr>
<td>5.2</td>
<td>Two methods of collecting transmissions with two different receivers to form datasets for testing hardware portability.</td>
<td>33</td>
</tr>
<tr>
<td>6.1</td>
<td>Visualization of the open-set metrics: AUROC, TPR, FPR.</td>
<td>39</td>
</tr>
<tr>
<td>6.2</td>
<td>Hardware Portability - Closed-set: Accuracy of models trained on data collected at (a)(b) RX1 and (c)(d) RX2 for each hardware portability dataset. N = 10% of training data, M = 10 inputs per decision.</td>
<td>41</td>
</tr>
<tr>
<td>6.3</td>
<td>Channel Portability - Closed-set: Accuracy of models trained on data collected on (a) Day 1 and (b) Day 2, when tested on data from five different days. N = 10% of training data, M = 10 inputs per decision.</td>
<td>46</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.4</td>
<td>Configuration Portability - Closed-set: Accuracy of models trained on data collected from transmitters using (a) Config. 1, (b) Config. 2, (c) Config. 3, and (d) Config. 4 when tested on data from transmitters using four different configurations. ( N = 10% ) of training data, ( M = 10 ) inputs per decision.</td>
<td>47</td>
</tr>
<tr>
<td>6.5</td>
<td>Configuration Portability - Multiple Calibrations - Closed-set: Accuracy of models trained on data collected from transmitters using Config. 2 when calibrated with data from multiple configurations. ( N = 10% ) of training data (per configuration), ( M = 10 ) points per decision.</td>
<td>48</td>
</tr>
<tr>
<td>6.6</td>
<td>Hardware Portability - Open-set: Open set performance of models trained on data collected at (a)-(f) RX1 and (g)-(l) RX2 for each hardware portability dataset. ( N = 10% ) of training data, ( M = 10 ) points per decision.</td>
<td>52</td>
</tr>
<tr>
<td>6.7</td>
<td>Channel Portability - Open-set: Open set performance of models trained on data collected on (a)(b)(c) Day 1 and (d)(e)(f) Day 2, and tested on data from five different days. ( N = 10% ) of training data, ( M = 10 ) points per decision.</td>
<td>58</td>
</tr>
<tr>
<td>6.8</td>
<td>Configuration Portability - Open-set: Open-set performance of models trained on data collected from transmitters using (a)(b)(c) Config. 1, (d)(e)(f) Config. 2, (g)(h)(i) Config. 3, and (j)(k)(l) Config. 4, and tested on data from transmitters using four different configurations. ( N = 10% ) of training data, ( M = 10 ) points per decision.</td>
<td>59</td>
</tr>
<tr>
<td>6.9</td>
<td>Configuration Portability - Multiple Calibrations - Open-set: Open-set performance of models trained on data collected from transmitters using Config. 2 when calibrated with data from multiple configurations. ( N = 10% ) of training data (per configuration), ( M = 10 ) points per decision.</td>
<td>60</td>
</tr>
<tr>
<td>6.10</td>
<td>Changing value of ( M ): Effect of changing the value of ( M ) in the decision making algorithm on accuracy of models from different evaluation scenarios: (a) hardware portability, (b) channel portability, and (c) configuration portability. The value of ( N ) was fixed to 10% for all data shown.</td>
<td>66</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.11</td>
<td>Changing value of ( N ): Effect of changing the value of ( N ) in the calibration algorithm on accuracy of models from different evaluation scenarios: (a) hardware portability, (b) channel portability, and (c) configuration portability. Note that ( M ) was fixed to 10 points per decision for all data shown. (Error bars represent 95% confidence interval based on 5 trials with random calibration data).</td>
<td>69</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Summary of the datasets used to evaluate performance in different portability scenarios.</td>
<td>30</td>
</tr>
<tr>
<td>5.2</td>
<td>Different LoRa transmitter configurations used in the experimental scenarios.</td>
<td>32</td>
</tr>
</tbody>
</table>
## LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Calibration</td>
<td>22</td>
</tr>
<tr>
<td>2  Closed-set Decision</td>
<td>24</td>
</tr>
<tr>
<td>3  Open-set Binary Decision</td>
<td>26</td>
</tr>
</tbody>
</table>
For Tyler, whose encouraging words and unending support are the only reason that this thesis was completed. I love you.
In the last few decades, the usage of electronic devices that emit RF (Radio Frequency) signals to communicate wirelessly has increased dramatically. Such RF devices are also being used for an increasing number of different applications, some of which deal with sensitive information. As such, security related to RF devices has become a research area of great interest.

Security for RF devices can be broken down into smaller aspects, each of which deals with different concerns. One aspect of security that is of great importance is authentication. If a communication system ensures authentication, then at some level it ensures that the identity of a sender and receiver are legitimate. That is, a receiver is assured that the message it is receiving is coming from a particular sender. Typical approaches to ensuring authentication involve the use of cryptographic systems, but these systems are constantly under threat from attackers. Given enough time and information, an attacker could break secure authentication and pose as someone they are not.

The concept of RF device fingerprinting seeks to provide a device-level solution to this issue of authentication (among other potential applications). RF device fingerprinting has been a very active area of research in the past several years, and it posits that: due to manufacturing tolerances, there will be small differences between the components of every RF transmitter (even of the same model). These
small differences manifest in the RF waves produced by these transmitters in a way that makes each transmitter uniquely identifiable. Thus, if a method could be identified to extract the information in the RF waves that identifies each transmitter, then this information could be used as a ‘fingerprint’ for each particular device. This could then be used to provide robust device-level authentication on top of existing layers of security for more secure wireless network access.

Several different approaches have been taken to attempt to achieve this type of RF device fingerprinting. Earlier approaches made use of hand-crafted features to extract particular parts of the RF transmissions of devices for further processing. However, most recent approaches have opted to make use of the concept of deep learning. These deep learning approaches are typically formulated as some form of supervised classifier. That is, they involve training a model with labeled data from RF transmitters (the training examples indicate which transmitter they belong to) and the goal of the model is to predict which transmitters produced some previously unseen test examples.

Approaches that apply basic deep learning on data captured from RF devices have been successful in performing accurate closed-set classification of these devices when the training data for the model closely matches with the testing data. However, it has become well understood that changing certain conditions for testing data negatively impacts classification accuracy for the same transmitters.

To demonstrate how changing some conditions for the testing data can impact accuracy, several tests were performed using data collected with the testbed of 25 LoRa transmitters and 2 USRP B210 receivers that will be fully introduced in
Chapter 5 of this work. Fig. 1.1a shows the accuracy of a deep learning model (not using the technique later proposed in this work) when it is tested with data captured at the same receiver as the training data (RX1) and a different receiver (RX2). It can be seen that performance is extremely poor when testing with a different receiver from that used to capture the training data, and thus the issue of hardware portability is demonstrated.

Fig. 1.1b shows the accuracy of the same deep learning architecture when it is trained with data captured on Day 1 and tested on data captured across five different days. This change in time represents a change in the time-varying wireless channel. It can be seen that model performance is poor when testing with data
collected on a day other than the training day. This demonstrates the issue of channel portability.

Finally, Fig. 1.1c shows the accuracy of this deep learning model when it is trained with data where the transmitters all use one configuration (LoRa spreading factor = 7) and is tested on data where the transmitters use four different configurations. The result is that performance is poor when the model is tested with data from transmitters using configurations other than the one used during training. This demonstrates the issue of configuration portability.

The results produced above are also demonstrated in other works [9, 19]. Since this problem is well established, developing a model that can overcome the aforementioned portability issues has become an active research area in recent years. Additional information concerning related works on portability will be presented later in Chapter 3. However, it is worth mentioning here some of the more naive approaches to addressing portability in order to clarify the alternatives available instead of using a technique such as the one proposed in this work.

One naive way to address portability is to train a new deep learning model for each domain (receiver, wireless channel, or configuration) that one wishes to test with. However, this approach is costly in terms of both the amount of labeled training data required from each new domain and the amount of computation required to train the models. Another naive alternative would be training a single deep learning model with data from multiple domains (receivers, wireless channels, configurations) in order to produce one model that is agnostic to these domains. This approach is also costly in terms of the amount of labeled training data required
from each domain, and this increased amount of training data also increases the computation time required to train the single model.

In contrast to such methods, this work proposes a lightweight domain adaptation technique, based on a combination of metric learning techniques and a calibration procedure, to solve these issues. By applying this technique, a deep learning model for RF device fingerprinting that is trained on data from one domain (receiver, day, configuration) can be calibrated to perform effectively on data from a different domain. This technique has a distinct advantage over conventional methods because this calibration process can be performed very quickly, without changing the model weights, and can be done using a relatively small amount of labeled data from the domain to which one wishes to calibrate the model. This makes this technique suitable for use cases in which computation resources are restricted.

Furthermore, the technique proposed in this work is easily adapted to act as an open-set classifier, which represents a more realistic classification scenario than a closed-set classifier, as it forces the model to deal with inputs from classes that are unknown. In brief, the main contributions of this work can be summarized as follows:

- **Lightweight model portability.** This work proposes a novel technique capable of producing a portable deep learning model that can be calibrated to perform RF fingerprinting on inputs from domains other than those represented by the training data. This technique makes use of metric learning to train the initial network and a quick calibration process which stores in-
formation about each device based on a small amount of examples from the target domain.

- **Multiple-domain model portability.** This work evaluates the ability of the proposed technique to produce a model that can be calibrated to perform RF fingerprinting in three different portability scenarios: hardware portability, channel portability, and configuration portability. This represents a more varied analysis than most related work.

- **Both closed-set and open-set identification.** This work evaluates the proposed technique in terms of both closed-set and open-set classification performance, while most related works that attempt to address portability do not perform open-set testing.

- **Real world dataset collection and evaluation.** This work evaluates the proposed technique using data collected from a testbed of 25 LoRa transmitters and 2 USRP B210 SDR receivers. Furthermore, 5 additional hardware portability datasets (involving two receivers) are collected for this work to complement the single hardware portability dataset created using this testbed in [8].

The remainder of this work is organized as follows: Chapter 2 outlines some potential use cases for the proposed technique. Chapter 3 briefly covers some related works on portability. Chapter 4 explains the details of the technique itself, and Chapters 5 and 6 cover evaluation of the technique with RF data captured using a real testbed of RF devices.
Chapter 2: Motivating Use Cases

We begin in this section by outlining some use case scenarios that illustrate the usefulness of the proposed technique in terms of its ability to significantly reduce (i) the training time and (ii) the amount of data that would have otherwise been needed to train additional neural networks.

2.1 Enabling Portability Across Different Receivers

Consider a scenario in which a model is trained using data collected at a receiver RX1, and suppose we wish to perform device classification on examples collected at another receiver RX2 that could, for instance, be located in a different physical location (e.g., another office building). As illustrated in Chapter 1, simply deploying the trained model from RX1 on RX2 and using it will not work and will achieve poor classification accuracy.

As shown in Fig. 2.1a, using the proposed technique, the user could simply first send/transfer the model trained on RX1 data to the new receiver RX2 and then calibrate it using a few samples captured at this new receiver. After calibration, the model deployed at RX2 will be able to perform effective classification on examples collected at RX2. Therefore, the proposed technique allows a ‘lightweight’ transfer of models across different hardware.
1. **Train** with RX1 data

2a. **Calibrate** with RX1 data and deploy at RX1 (optional)

2b. **Send** trained model

3. **Calibrate** with RX2 data and deploy at RX2

(a) Overcoming Receiver Hardware Variability

---

Day 1

1. **Train** with Day 1 data

2a. **Calibrate** with Day 1 data and deploy on Day 1 (optional)

(b) Overcoming Wireless Channel Variability

---

Day 2

3. **Calibrate** model with Day 2 data and deploy on Day 2

---

1. **Train** with Config. 1 data

2a. **Calibrate** with Config. 1 data and deploy (optional)

2b. **Collect** data from additional Configs.

3. **Calibrate** with data from multiple Configs. and deploy

(c) Overcoming Protocol Configuration Variability

Figure 2.1: Transferring of a trained model to a different domain (hardware, channel, configuration) via calibration.
2.2 Enabling Portability Across Different Channels

Another use case arises from the scenario in which a model is trained using data collected during one day, and we wish to continue using the trained model for classification on subsequent days. As was shown in Fig. 1.1b, simply using the model trained with Day 1 data to perform classification on data from some other day will not work, due to the changes that occur in the wireless channel over time.

To overcome this issue, we leverage the proposed technique as shown in Fig. 2.1b, and collect a small amount of additional labeled examples after some time. The existing model is then re-calibrated using the collected examples. With this lightweight calibration, the proposed technique enables the model to perform effectively on data collected at different times (i.e., different wireless channels) without needing to re-train the model every time.

2.3 Enabling Portability Across Multiple Configurations

Consider a scenario wherein a model is trained on data collected when the transmitters are using one LoRa configuration (e.g., a LoRa spreading factor of 7), and we wish to classify these same transmitters even when they are using a different LoRa configuration. As discussed in Chapter 1 via Fig. 1.1c, such a trained model will not perform well when transmitters change their protocol configurations, thereby giving rise to an important issue, as LoRa is designed to allow transmitters to change their configurations to adapt to network conditions (e.g., transmit power, channel condition, data rates, etc) [12, 3]. This means that a model trained with
one configuration may fail in a real world scenario when a transmitter switches configurations.

The proposed technique, shown in Fig. 2.1c, addresses this issue, and does so by collecting a small amount of examples from each transmitter using all possible configurations, and calibrating the model using these examples. The resulting model is able to classify examples from the transmitters using any one of these configurations.
Chapter 3: Related Work

Much work has already been done within the research area of RF device fingerprinting. As such, there are a variety of works that make use of deep learning methods to perform closed-set and open-set classification of RF transmitters. Instead of referencing all recent work done on RF fingerprinting in general, this section will highlight those works that have focused on the problem of portability in RF fingerprinting. These works can be categorized by the type of portability that they seek to address.

3.1 Works Addressing Channel Portability

The methods proposed in [20, 2, 14, 18, 22] all seek to overcome channel portability issues. That is, they seek to produce RF fingerprinting models that perform well across different wireless channels due to either variations in time or physical location. Of these works, [2, 14, 18, 22] aim to make use of data augmentation techniques and/or custom feature extraction methods in order to obtain an RF fingerprint from an initial training dataset that is theoretically invariant to changes in channel conditions. That is, these works propose methods that do not rely on labeled data from a target domain to adjust a model, but instead seek to train an initial model that performs well regardless of the channel conditions of input data.
This approach, while effective for certain conditions, produces a rigid model that may not be adapted to differing conditions later, and thus may require re-training.

In contrast to these methods, the authors in [20] propose a technique for creating a system that can be adjusted to perform well on different wireless channels than the one used for training, given some labeled data from the target channel. Thus, the approach in [20] is more similar to the one taken in this work. A downside of the technique in [20], however, is that it requires transmitters to implement an FIR filter as specified by the receiver which may not be possible in some use cases.

3.2 Works Addressing Hardware Portability

The methods proposed in [16, 4, 28] all seek to address the issue of hardware portability. That is, they seek to produce RF fingerprinting models that can perform classification when different receiver hardware is used to capture the input transmissions than was used to capture training data. To accomplish this, the authors in [16] propose the use of a ‘trainable transform network’ which can be trained as part of a larger model using data from several transmitters captured at multiple receivers to eliminate the effect of the receivers during testing. The only notable downside of this technique is that a learning process must be done with labeled data in order to add additional receivers.

The authors of [28] propose something similar to [16], where the idea is to reduce the distance between the distribution of the data collected at a target receiver
and the data collected at the receiver used to obtain training data, however the
publication that could be found lacked details. The authors of [4] also use a similar
idea to [16], but use transmissions from a single transmitter captured at multiple
receivers to extrapolate the differences between the receivers. A function to remove
any receiver bias is then applied to new inputs. One downside of the method used
in [4] is that it lacks generality, since it is based on transmissions from a single
device.

3.3 Works Addressing Multiple Types of Portability

The methods proposed in [21, 17] address multiple types of portability. The authors
of [17] develop a method that seeks to address channel portability and receiver
hardware portability. This method falls into the category of methods that seek
to use data augmentation and customized feature extraction to produce an RF
fingerprint that is truly portable without the need for labeled data from any target
domain. In this sense, it has the same disadvantage mentioned earlier: it produces
a model that is rigid and requires re-training for any receivers or channels is was
not trained to work with.

In [21] the authors develop a technique that seeks to address both channel
portability and another type of portability not addressed by any other work men-
tioned: transmitter hardware portability. That is, this method seeks to produce
a model capable of being trained using one set of transmitters and deployed to
perform classification on another set of transmitters. It is also notable that the
work in [21] uses a method that is very similar to the one used in this work in the following ways: it makes use of metric learning, it uses the triplet loss function for network optimization, and it uses an ‘enrollment’ processes which is similar to the ‘calibration’ process described in this work.

This work differs from [21] in several ways though: k-NN (k Nearest Neighbor) is used as backbone of the decision-making method in [21] and instead of explicitly adapting a trained model to different wireless channels, the authors in [21] attempt to extract an RF fingerprint that is channel agnostic in the first place. Due to its similarity, it can be postulated that the technique proposed in this work could be scalable in the same sense that the work in [21] is, and thus it could be used to achieve transmitter hardware portability as well (note: most related works do not achieve scalability since they often use a softmax output on their neural networks, which indicates some form of re-training would be required to alter the number of transmitters the network could classify). It is also notable that, to the best of our knowledge, the work in [21] is the only one of all those mentioned that also addresses the open-set problem.

In short, most related works focus on a single aspect of portability and do not address the open-set problem. In contrast, this work does address the open-set problem, and focuses on channel portability, hardware portability, and configuration portability (no closely related work could be found on configuration portability). It is also worth mentioning that most of the related works do not focus on the LoRa protocol, with only the work in [2, 21, 18] using LoRa data.
Chapter 4: Domain-Agnostic Device Fingerprinting

Our goal is to provide a method by which a closed-set or open-set deep learning model can be trained on data from one domain (collected at one receiver, through one wireless channel, or with transmitters using one configuration) and then be calibrated to perform well on data from a different but related domain. In this way, the proposed technique aims to move toward the creation of deep learning models for RF fingerprinting that are portable in terms of hardware, wireless channel, and transmitter configuration.

These portability goals will be achieved while using a calibration process that is: 1) not computationally intensive (and can be done on hardware that is less powerful than the resources used for training the original model) and 2) able to be accomplished with a limited amount of labeled training data from the target domain (receiver, day, configuration).

Fig. 4.1 summarizes the proposed technique. Prior to deployment, this tech-

![Figure 4.1: A high-level diagram of the proposed technique](image-url)
nique requires a training phase for the twin neural network and a calibration pro-
cess using data from the target domain from all devices. Note that the calibration
process can be repeated with different data without altering the twin neural net-
work. The following subsections provide more information including background
material on the twin network architecture, details regarding how the neural network
is trained as well as specifics of the calibration and decision-making algorithms.

4.1 Deep Learning

Most recent techniques for RF device fingerprinting make use of deep learning. A
comprehensive treatment on deep learning is outside the scope of this work and
additional information can be found in [11]. This section will focus the discussion
on metric learning and twin neural networks. We will also be using the twin neural
networks as closed-set and open-set classifiers. Closed-set classifiers are designed
to classify data with the assumption that they only encounter data instances from
the classes observed during training. Open-set classifiers, on the other hand, can
handle data from classes that are not seen during training. There are many ways
to deal with data instances from these unseen classes. The approach that we will
take, which is commonly used in the open-set literature, is to identify them as
not belonging to one of the known classes. While the open-set problem is more
difficult, it is also more realistic as there are many cases in which a model deployed
in the real world will encounter data instances from outside its known classes.
4.1.1 Metric Learning and Twin Neural Networks

The standard approach for building a DNN classifier is to train the model using a cross entropy loss with one output node for each class. The values of these output nodes are then used to determine the degree to which an input is predicted to belong to a particular class. Under this standard supervised learning setting, it is non-trivial to adapt the network to different domains such as a different receiver, day or configuration. Sophisticated domain adaptation methods for deep learning (e.g. [26, 25]) have been developed, but these methods are typically computationally intensive and require large amounts of data. In contrast, we want a lightweight domain adaptation process that can be quickly calibrated from a small number of labeled examples from the target domain.

In order to overcome these deficiencies, our work makes use of deep metric learning [10] in which a DNN produces an embedding of a data instance in a \( K \)-dimensional latent space. Within this embedding space, data instances that are similar (e.g. from the same device) will be closer than those that are not similar (e.g. from different devices). The proposed technique will use a well known metric learning structure called a twin neural network (aka ‘Siamese’ neural network) [5, 6]. These types of neural networks have been successfully used for applications where few examples are available such as signature verification and facial similarity calculations [27, 7]. They have also been used for classification of RF signals based on spectrogram images [15]. Additionally, it has been shown that twin networks trained on one image dataset and evaluated on an entirely different image dataset
A general twin neural network architecture using the triplet loss function during training.

(e.g., trained on Omniglot and evaluated on MNIST) are able to generalize and maintain some level of performance [13]. However, they have only been applied directly to the problem of device fingerprinting based on RF signals in one other work [21].

A diagram representing a twin neural network during training is shown in Fig. 4.2. Twin networks consist of a pair of DNNs with identical weights designed to take one input example each. The output nodes of these networks can each be treated as the coordinates of a point in some $K$-dimensional latent space ($K$ being the number of output nodes). Thus, each input will produce a single point in this latent space. The distance between the outputs in the latent space is computed as a measure of similarity and can be used to train the network with a loss function or to make decisions regarding the inputs once the network has been trained. It is also of note that in practice a twin network can be considered as a single neural network that produces an output point, instead of twin networks that produce a distance (since the networks have identical weights).
In order to train a twin neural network, it is desirable to use a loss function that will cause the outputs produced by each input to be less similar for inputs from different classes and more similar for inputs from the same class. This will allow the trained network to make decisions about whether inputs came from the same class or not by measuring the distance between (similarity of) the outputs produced.

To achieve this goal, the well-known triplet loss function often used with twin neural networks can be used [24, 23]. During training, this loss function encourages reduced distances between ‘anchors’ and ‘positive’ examples, and increased distances between ‘anchors’ and ‘negative’ examples. The mathematical expression for this loss function is provided in Equation (4.1) and a visual depiction is provided in Fig. 4.3. In both the figure and equation: A represents an ‘anchor’ example, P represents a ‘positive’ example from the same class as A, and N represents a ‘negative’ example from a different class than A, \( \alpha \) represents a margin value (indicating the loss value when \( P \) and \( N \) are the same distance from \( A \)), and \( f(\cdot) \) represents the neural network mapping function.

\[
Loss = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0) \tag{4.1}
\]
When using triplet loss, triplets of ‘anchor’, ‘positive’, and ‘negative’ examples must be provided to the network during training. There are multiple ways to create these triplets but two dominant approaches are: to generate them from training data before any examples have passed through the network, or to generate them after examples from a mini-batch have already passed through the network but before loss is calculated. It has been shown that it is advantageous to determine triplets after examples have passed through the network because this way more difficult triplets can be “mined” [23]. This “mining” involves selecting triplets that have $A$ closer to $N$ than to $P$ (high loss triplets) purposefully in order to help the network learn to separate these examples in the next gradient update.

4.1.2 Training the Twin Neural Network

To achieve portability in an RF fingerprinting model, we train a twin network using the triplet loss function for the proposed technique shown in Fig. 4.1.

Once this model has been trained on some labeled data from a chosen domain (receiver, channel, configuration), inputs from different transmitters should be mapped to outputs that are farther apart, and inputs from the same transmitter should be mapped to outputs that are closer together. In this state, the model is typically only capable of making decisions given two inputs that it can compare in terms of their distance in the latent space. In order to enable the model to make decisions regarding a single input, a different decision making scheme must be developed.
4.2 Calibration and Decision Making Algorithms

To enable the trained twin network to make decisions about a single input, a calibration task must first be performed, and then a decision making task can be performed.

4.2.1 Calibration Task

After training, the first requirement is to calibrate the trained twin neural network to make decisions about data from a particular domain. This will be done by providing the trained network with labeled examples from all desired classes. Note that these examples may come from the training data if calibrating for the same domain as the training data.

The calibration process is described in Algorithm 1 and illustrated in Fig. 4.4. For each class (Device), we repeat the following process: given $N$ examples from class $i$ for calibration, $x_{i1}, x_{i2}, \ldots, x_{iN}$, we first pass these examples through the
Algorithm 1 Calibration

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Centroids} = []
\State \textbf{Distances} = []
\State \textbf{Devices} = \{D_1, D_2, ..., D_K\}; each D_i = \{x_{i1}, x_{i2}, ..., x_{iN}\}
\ForAll{D_i \in \textbf{Devices}}
\State \textbf{Centroids}[i] = \sum_{j=1}^{N} f(x_{ij}) / N
\State \textbf{Distances}[i] = \sum_{j=1}^{N} \|f(x_{ij}) - \textbf{Centroids}[i]\| / N
\EndFor
\end{algorithmic}
\caption{Calibrated for \(D_i \in \textbf{Devices}\)}
\end{algorithm}

trained neural network, \(f(\cdot)\), to produce \(N\) corresponding outputs in the latent space. The first image in Fig. 4.4 shows the result of this step. Next, these \(N\) outputs are averaged to produce the Centroid for class \(i\), which is shown in the second image of Fig. 4.4. This Centroid is the first piece of calibration data that will be stored for class \(i\).

Next, the distances between the Centroid for class \(i\) and each of the \(N\) outputs that were used to generate the Centroid are calculated. This is depicted in the third image of Fig. 4.4. These distances are averaged to produce the Distance for class \(i\), which is represented by a circle with this Distance value as its radius in Fig. 4.4. This Distance is the second piece of calibration data that will be stored for class \(i\). After the above process is repeated for every class, calibration is complete and the Centroids and Distances for the classes can be used to make decisions regarding new inputs using algorithms that will be described later.

This calibration process is designed to be lightweight, as it does not modify the trained weights of the twin neural network in any way and is only as computationally complex as passing examples forward through the network and performing
distance calculations. Additionally, it is notable that the value of N in this algorithm represents the number of examples from each device used for calibration purposes, and is a parameter that can be changed to effect the amount of computations required for calibration as well as the amount of labeled data required.

Once a trained network has been calibrated with data from a particular domain and set of devices (e.g. data from devices 1-10 captured using receiver 1), it is ready to make decisions about inputs that come from the same devices and domain. It is also of note that the calibration process could be repeated with data from multiple domains for the same devices in order to produce a model capable of making decisions about data from these multiple domains (note that this would produce multiple Centroids and Distances for each device).

4.2.2 Closed-set Decision Making Task

To make closed-set classification decisions, the calibrated network follows Al-
Algorithm 2. This algorithm attempts to predict the device that transmitted the input data from among the set of known devices. To begin, $M$ input examples, $x_1, x_2, ..., x_M$, are collected from an input device and passed through the trained neural network, $f(\cdot)$, to produce $M$ output points. These $M$ output points are then averaged together to produce the Input Point for the decision making process.

Since the aforementioned operation is not typical in deep learning models, it deserves some explanation. Generally, this type of processing of multiple inputs is not considered to be possible in other domains in which DNNs are applied (e.g. computer vision). However, the RF domain is unique in the sense that the examples networks are trained to operate on represent fractions of a second of real-time RF communication. Thus, it is plausible for many applications of RF fingerprinting to assume that $M$ examples could be collected for the model to make each decision. This operation is considered to be advantageous because it allows for a less noisy
input to be constructed that is more representative of a particular device and less susceptible to the effects of outlier examples.

After generating the *Input Point*, two quantities are then computed for each class using this point and the calibration data (*Centroids* and *Distances*) previously generated for each class $i$:

- $A_i$: the distance between the *Input Point* and $Centroids[i]$ in the latent space.

- $B_i$: the difference between $A_i$ and $Distances[i]$.

The calculation of $A_i$ and $B_i$ is illustrated by Fig. 4.5. The left image in this figure shows the case where $B_i$ is positive, and the right image shows the case where it is negative. These images make it clear that $B_i$ will be negative if the distance between the *Input Point* and $Centroids[i]$ is less than $Distances[i]$, and that $B_i$ will be non-negative otherwise.

After calculating $A_i$ and $B_i$ for every class $i$, the algorithm then checks the $B_i$ values. If all $B_i$ values are non-negative, this indicates that the distance from the *Input Point* to $Centroids[i]$ was greater than $Distances[i]$ for every class $i$, and so the algorithm returns the decision of the class $i$ with the minimum $B_i$ value (the class where the distance from the *Input Point* to the Centroid was least in excess of the Distance).

If any of the $B_i$ values are negative, then the algorithm returns the decision of the class $i$ with the minimum $A_i$ value among those classes that had a negative $B_i$.
value (the class with minimum distance between the Input Point and its Centroid, as long as this distance is less than the Distance).

4.2.3 Open-set Decision Making Task

Algorithm 3 Open-set Binary Decision

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input Device Examples} = \{x_1, x_2, ..., x_M\}
\State \textbf{Input Point} = \frac{\sum_{j=1}^{M} f(x_j)}{M}
\For{\text{all } D_i \in \text{Devices}}
\If{dist(\text{Input Point}, \text{Centroids}[i]) \leq \text{Distances}[i]}\text{Return: Decision = Admit}\EndIf
\EndFor
\State \text{Return: Decision = Reject}
\end{algorithmic}
\end{algorithm}

To make open-set classification decisions, the calibrated network follows Algorithm 3. This algorithm makes a binary decision on open-set inputs, and seeks to determine if an input originates from a known class or not. Similar to the closed-set decision algorithm, the open-set algorithm begins by collecting $M$ examples, $x_1, x_2, ..., x_M$, from an input device, passing these examples through the trained network, $f(\cdot)$, and averaging the $M$ outputs to produce an Input Point.

After the Input Point is computed, for each class $i$ the algorithm calculates the distance between the Input Point and Centroids[$i$] and compares this distance with Distances[$i$]. If at any point during this process, this calculated distance is less than or equal to Distances[$i$], the algorithm decides that the input comes from a known class, and returns the ‘Admit’ decision. If none of the calculated distances
are less than or equal to the corresponding Distance, then the algorithm decides the input came from an unknown class, and returns the 'Reject' decision. ('Admit' and 'Reject' corresponding to a use case where this model is used as some means of authentication).
Chapter 5: Experimental Scenarios and Dataset Collection

This section first covers the details of the testbed of devices that was used to capture all of the RF data utilized in this work. The second half of this section then details the specific datasets collected to evaluate the proposed technique under three different portability scenarios.

5.1 Testbed

In order to evaluate the proposed technique, real RF data was collected using a testbed of devices. This testbed contains 25 almost identical PyCom IoT devices used as transmitters: 23 Lopy4 boards and 2 Fipy boards on top of 22 Pysense sensor shields, 2 Pytrack sensor shields, and 1 Pyscan sensor shield (pictured in Fig. 5.1c). The testbed also uses 2 USRP B210 SDRs (Software Defined Radios) as receivers to collect data (pictured in Fig. 5.1b).

All of the datasets collected using this testbed (outlined in the next section) use the following settings and processes unless otherwise stated: the PyCom transmitters were configured to transmit using the LoRa protocol, with a center frequency of 915 MHz, spreading factor of 7, and bandwidth of 125KHz. The USRP B210 receivers were configured to sample at a center frequency of 915 MHz, at a rate of 1M samples per second. Each transmission sends the same message, and lasts for 20s.
Figure 5.1: (a) One of the two identical halves of the twin neural network architecture. (b) and (c) images of testbed hardware.
Table 5.1: Summary of the datasets used to evaluate performance in different portability scenarios.

<table>
<thead>
<tr>
<th>Portability</th>
<th>Env.</th>
<th>Multi-RX Collection</th>
<th># TX</th>
<th># RX</th>
<th># Days</th>
<th># TX Config.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Indoor</td>
<td>Same Tx</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Hardware</td>
<td>Outdoor</td>
<td>Same Tx</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Hardware</td>
<td>Wired</td>
<td>Same Tx</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Hardware</td>
<td>Indoor</td>
<td>Diff. Tx</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Hardware</td>
<td>Outdoor</td>
<td>Diff. Tx</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Hardware</td>
<td>Wired</td>
<td>Diff. Tx</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Channel</td>
<td>Outdoor</td>
<td>N/A</td>
<td>25</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Config.</td>
<td>Indoor</td>
<td>N/A</td>
<td>25</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

This produces 20M complex-valued samples for each transmission received. This data was stored in raw-IQ format and FFT format, using GNURadio to process the data (note that only raw-IQ data is used in this work).

Regardless of the environment in which data was collected, the transmitters and receivers were always placed about 5m apart (with the exception of ‘Wired’ data, which is collected by connecting transmitters to receivers by physical wire). All data collected using this testbed has been or will be made publicly available as soon as possible. Additional information about the testbed can be found in [8], which is the work that describes the first datasets this testbed was used to collect.

5.2 Experimental Scenarios

Several datasets were collected using the testbed in order to test the performance of the technique in various portability scenarios, including: hardware portability, channel portability, and transmitter configuration portability. A summary of these
datasets is provided in Table 5.1.

**Hardware portability dataset scenario.** There are six different datasets that were collected to evaluate the hardware portability scenario that can be seen in Table 5.1. All of these datasets involve the same 2 receivers and 25 transmitters, and each contains 2 recorded transmissions from each transmitter: one at each receiver.

These six datasets differ in terms of the environment they were collected in as well as the ‘Multi-RX Collection’ method used. Datasets where the ‘Multi-RX Collection’ method is ‘Diff. Tx’ were collected by capturing two different transmissions from each device, one at each receiver as shown in Fig. 5.2b. Datasets where this method is ‘Same Tx’ were collected by capturing the same single transmission from each device at both receivers as shown in Fig. 5.2a. Intuitively, the ‘Same Tx’ datasets represent cases where it was originally postulated that it could be easier for a deep learning model to adjust to performing classification on data from the opposite receiver, since the transmission was literally the same and not just from the same transmitter with the same contents.

**Channel portability dataset scenario.** There is a single dataset to evaluate the channel portability scenario in Table 5.1. This dataset includes 5 transmissions from each of the 25 transmitters, each of the 5 transmissions collected on a different day. Since the wireless channel changes over time, this dataset represents transmissions from the transmitters with 5 different wireless channel conditions.

**Protocol configuration portability dataset scenario.** Finally, there is a single dataset to evaluate the configuration portability scenario in Table 5.1.
This dataset includes four transmissions from each of 25 transmitters, each of the four transmissions using a different configuration. The configurations amount to using different LoRa spreading factors and are detailed in Table 5.2. Note that all other datasets use only Config 1. (note: specific configurations will be referred to as Configs. from here on for brevity).

It is also worth mentioning that some of the datasets used for evaluation in this work had been previously collected using the testbed and released in another work, where they are referred to by different names [8]. Specifically, the hardware portability dataset in the ‘Indoor’ environment using the ‘Diff. Tx’ method is the ‘Different Receivers Scenario’ dataset, the channel portability dataset is part of the ‘Different Days Outdoor Scenario’ dataset, and the configuration portability dataset is the ‘Different Configurations Scenario’ dataset [8].

<table>
<thead>
<tr>
<th>Config.</th>
<th>Spreading Factor</th>
<th>BW</th>
<th>Tx Power</th>
<th>Coding Rate</th>
<th>Bit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>125kHz</td>
<td>20dBm</td>
<td>4/5</td>
<td>5470bps</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>125kHz</td>
<td>20dBm</td>
<td>4/5</td>
<td>3125bps</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>125kHz</td>
<td>20dBm</td>
<td>4/5</td>
<td>537bps</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>125kHz</td>
<td>20dBm</td>
<td>4/5</td>
<td>293bps</td>
</tr>
</tbody>
</table>

Table 5.2: Different LoRa transmitter configurations used in the experimental scenarios.
(a) Same Transmission

Each TX transmits once, and transmission is received by both RX1 and RX2

(b) Different Transmission

Each TX transmits twice, once for RX1 and once for RX2

Figure 5.2: Two methods of collecting transmissions with two different receivers to form datasets for testing hardware portability.
Chapter 6: Experimental Results

This section covers the experimental results of this work by first describing both the specific implementation of the twin network architecture as well as the metrics that will be used to evaluate the technique. The bulk of this section then explains the results obtained by testing the proposed technique under the portability scenarios outlined in the previous section. Finally, the effects of some variables present in the calibration and decision-making algorithms are explained and the computational speed of the technique is made clear.

6.1 Neural Network Implementation

In order to evaluate the proposed technique, the twin neural network described earlier had to be implemented. To accomplish this, the popular Python library ‘PyTorch’ was used to define, train, and test the network.

The network used for this work is inspired by one of the straightforward convolutional networks described in [1] and is depicted in Fig. 5.1a. Note that this represents one of the two identical halves of the twin network, which are in practice the same network. The network begins with two 1-D convolutional layers which have 128 filters each, and kernel sizes of 7 and 5 respectively. These convolutional layers are followed by a 1-D max-pool layer and a 1-D batch normalization layer.
This is followed by an additional two 1-D convolutional layers which have 256 filters each, and kernel sizes of 7 and 5 respectively. These layers are followed by another 1-D max-pool layer and another 1-D batch normalization layer. The network ends with two fully-connected layers, which have 256 neurons and 12 neurons respectively. This produces a 12-dimensional output point for each input example. It is also notable that Leaky ReLU is used as the activation function throughout the network.

The input data for RF fingerprinting is typically composed of In-phase (I) and Quadrature (Q) samples taken from RF transmissions. This makes sense as it is common for signals to be represented in the form of complex IQ data, where I is the real part of a complex number and Q is the imaginary part. For the purposes of this work, a sequence length of 128 complex samples was considered as one input example. Additionally, two different input channels were considered, one for I samples and one for Q samples. Thus, the final input shape for the network is 2x128.

The triplet loss function is used to train the network with a margin value of 0.1, and the hard negative “mining” strategy described earlier is employed to select triplets for training. The optimizer chosen was standard SGD (Stochastic Gradient Descent) with Momentum set to 0.9. Batch size was set to 64, and the learning rate for each model was tuned by performing a search in the typical range of 0.01 to 0.000001 to ensure decreasing loss on the training set. The networks were allowed to train for 100 epochs and the model was saved at the best performing epoch. After training was completed, calibration was performed on the desired dataset.
according to the method described by Algorithm 1 (the data used for calibration was changed depending on the evaluation).

In addition to defining a network using the proposed technique, a ‘vanilla’ network was defined using a similar architecture to the one shown in Fig. 5.1a, for comparison purposes. The structure of this ‘vanilla’ network is exactly the same as the network for the proposed technique, except that the number of neurons in the final layer is 10 (as 10 transmitters will be used for classification). The same hyper-parameters described for the proposed network are used for training, but the ‘vanilla’ network makes use of cross-entropy loss instead of triplet-loss.

6.2 Performance Evaluation Metrics

6.2.1 Closed-set metrics

To evaluate trained models on closed-set data (examples only from classes the model was trained on), a basic closed-set accuracy metric was used. This accuracy metric is calculated as the number of correct predictions (model prediction matches class of input) over the total number of predictions.

6.2.2 Open-set metrics

To evaluate trained models on open-set data (examples may be from the classes the model was trained on or from unknown classes), three different metrics are used: (i) the averaged Area Under the Receiver Operating Characteristic (Avg. AUROC);
(ii) the averaged True Positive Rate (Avg. TPR); and (iii) the averaged False Positive Rate (Avg. FPR). These three metrics are further explained next.

The AUROC metric allows for measuring the performance of a binary classifier (i.e., classifying an input/device as known or unknown) without having to specify a particular threshold. This means that this metric reflects more about the usefulness of the metric learning technique for performing open-set binary classification and less about the particular choice of threshold. This metric enabled comparison of the open-set performance of ‘vanilla’ models and models created using the proposed technique without having to empirically tune the threshold for the models. This also means that the open-set evaluations using AUROC do not strictly follow Algorithm 3 defined earlier, as this algorithm uses specific thresholds (the Distances). In contrast, the TPR and FPR metrics are used to evaluate open-set performance using the exact open-set decision making process defined in Algorithm 3, requiring a particular threshold (the Distance) for each device.

Instead of using a particular threshold, AUROC evaluates all possible thresholds after all test examples has been given a decision value by the trained model. In the case of ‘vanilla’ models that make use of cross-entropy loss, this decision value will be set as the maximum logit value for each example. This is a simple way to adapt a typical closed-set classifier to operate on the open-set, and it means that examples producing lower than a certain maximum logit threshold will be rejected, and those above accepted.

In the case of the models created using the proposed technique, the minimum distance to a Centroid will be used as the decision value. This means that examples
whose distances are higher than some threshold will be rejected, and those whose
distances are lower will be accepted.

When evaluating all possible thresholds, two quantities, $TPR = TP/(TP + FN)$
and $FPR = FP/(FP + TN)$, are calculated for each threshold value, where

- True Positive (TP) is the number of tested known devices that the model
  predicts as known (i.e., Admitted by the trained model rightly)

- False Negative (FN) is the number of tested known devices that the model
  predicts as unknown (i.e., Rejected by the trained model wrongly)

- False Positive (FP) is the number of tested unknown devices that the model
  predicts as known (i.e., Admitted by the trained model wrongly)

- True Negative (TN) is the number of tested unknown devices that the model
  predicts as unknown (i.e., Rejected by the trained model rightly)

These quantities are also shown visually in Fig. 6.1b. Note that TPR and FPR
are also used for calculating open-set performances with Algorithm 3 when using
thresholds (the Distances).

After TPR and FPR are calculated for all possible thresholds, a point is plotted
on a curve for each threshold with TPR being the y-coordinate and FPR being the
x-coordinate. This forms the well-known ROC (Receiver Operating Characteris-
tic), an example of which is shown in Fig. 6.1a. The ideal classifier achieves an
ROC that touches the top-left corner of the axes, achieving a TPR of 1 and FPR
of 0 for some threshold. In order to quickly summarize this curve, the area under-
neath it can be calculated, which yields the AUROC metric. Since the maximum
39

(a) Example of a ROC curve, the area under which defines the AU-ROC metric

(b) The possible results of a single binary open-set decision, which are used to calculate TPR and FPR

Figure 6.1: Visualization of the open-set metrics: AUROC, TPR, FPR.

area under the curve is 1, a perfect AUROC value is 1, and the AUROC value of a random binary classifier is 0.5.

The averages of all of the open-set metrics were taken because the performance of the open-set models depends on which unknown classes are used. So, for each open-set evaluation of a model, 5 trials were performed. For each trial, an equal number of examples were drawn from 5 random known devices and 5 random unknown devices. Then the AUROC, TPR, and FPR results for each trial were averaged to produce the final values.

6.3 Closed-set Evaluation

In this section, models will be evaluated on the closed-set of devices that they were trained on. All models were trained and tested using data from the same 10
wireless transmitters, where 75% of the data from a single transmission from each device was used for training, and 25% was reserved for testing (each transmission contains 156,250 examples of size 128x2). In all tests, unless otherwise stated, the amount of calibration data used to perform the proposed technique \((N\text{ in Algorithm 1})\) was set to 10% of the size of the training data used to initially train the model and the number of input examples used to form the ‘input point’ \((M\text{ in Algorithm 2})\) was set to 10.

For comparison to the proposed technique, ‘vanilla’ models representing basic closed-set classifiers are also evaluated in this section. These ‘vanilla’ models use almost the same DNN structure as the proposed technique (shown in Fig. 5.1a), but are trained using the more typical cross-entropy loss and do not undergo the calibration process or use the defined decision algorithms. Otherwise, they are trained using the same data and parameters as the twin network models that implement the proposed technique.

### 6.3.1 Hardware Portability

To evaluate hardware portability of the proposed technique, each of the six hardware portability datasets listed in Table 5.1 was used to train two twin network models: one using only the RX1 data and one using only the RX2 data. Each of these trained models was then tested with data from both receivers under two conditions: when calibrated with data from the same receiver as training, and when calibrated with data from the receiver not used for training.
<table>
<thead>
<tr>
<th>Calibration Data</th>
<th>Indoor/RX1</th>
<th>Outdoor/RX1</th>
<th>Wired/RX1</th>
<th>Indoor/RX2</th>
<th>Outdoor/RX2</th>
<th>Wired/RX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.2: Hardware Portability - Closed-set: Accuracy of models trained on data collected at (a)(b) RX1 and (c)(d) RX2 for each hardware portability dataset. \( N = 10\% \) of training data, \( M = 10 \) inputs per decision.
Figs. 6.2a and 6.2b show the results of these evaluations for the models trained using RX1 data, and Figs. 6.2c and 6.2d show the results of the same evaluations for the models trained using RX2 data. In all sub-figures of Fig. 6.2, the results of training and testing ‘vanilla’ models on the same data are also shown for comparison.

Some trends are immediately clear by observing these figures. First, it can be seen that regardless of the environment of the dataset (Indoors, Outdoors, Wired), a model that is trained and calibrated with data from the same receiver (e.g. trained and calibrated with RX1 data) shows similar performance to a ‘vanilla’ model that is trained on data from that receiver. For instance, this can be seen in Figs. 6.2a and 6.2b by observing the similarity of the RX1 tests for the ‘vanilla’ model (first three bars) and the RX1 tests where the proposed model is calibrated with RX1 data. This indicates that using the proposed technique without attempting to achieve portability does not diminish performance.

More interesting results can be seen by observing the tests where the proposed model is calibrated with RX2 data in Figs. 6.2a and 6.2b. These tests represent using the proposed technique to achieve hardware portability by calibrating for a receiver other than the one used to capture training data. The ideal situation would be that these tests produce results similar to those achieved by a model trained using data from the opposite receiver and evaluated on the same data. For instance, it would be desirable for the results achieved when calibrating with RX2 data in Fig. 6.2a to look very similar to the results achieved when calibrating with RX2 data in Fig. 6.2c. When this desirable result occurs, it indicates that
the proposed technique is achieving some level of hardware portability, as a model trained using data from one receiver was able to perform well when tested with data from another receiver.

The result described above can be observed in all cases where the environment of the dataset is ‘Indoors’ or ‘Outdoors’. However, when the environment where the data was collected is ‘Wired’, this result does not occur reliably. Instead, it can be seen that some models trained in the ‘Wired’ environment are unable to be effectively calibrated to the opposite receiver. This can be seen in Fig. 6.2b where the Wired/RX2 test for the model calibrated with RX2 data does not achieve performance comparable to that of the Wired/RX2 test for the model calibrated with RX2 data in Fig. 6.2d. This can be seen in all cases where a model trained using ‘Wired’ data is calibrated for the opposite receiver from the one used to train the model.

It should, however, be noted that even in the cases where the proposed technique is unable to achieve the desired result as described above, it does result in better performance than a model trained using data from one receiver blindly tested on data from another receiver. For instance, in Fig 6.2b, the accuracy of the model trained with Wired data when it is calibrated and tested on RX2 data is roughly double the accuracy of the same model when calibrated with RX1 data and tested on RX2 data.
6.3.2 Channel Portability

To evaluate the channel portability of the proposed technique, the channel portability dataset listed in Table 5.1 was used to train two twin network models: one using only data collected on Day 1 and one using only data collected on Day 2. Each of these trained models was then calibrated for each of the five days in the dataset. Finally, each possible calibrated model was tested with data from all five days.

Fig. 6.3a shows the results of these tests for the model trained with data collected on Day 1, and Fig. 6.3b shows these results for the model trained with data collected on Day 2. Note that in all sub-figures of Fig. 6.3, the results of training and testing a ‘vanilla’ model on the same data are also shown.

The trend that is immediately clear in both of these figures is that when a model is calibrated with data from a particular day, it performs very well when tested on data from that same day. For instance, in Fig. 6.3a it can be seen that the model trained with Day 1 data and calibrated with Day 3 data performs very well (> 90% accuracy) when tested with Day 3 data, and very poorly (< 40% accuracy) when tested with data from other days. This trend is consistent across all tests in Fig. 6.3, regardless of the day to which the model is calibrated. In fact, the lowest accuracy achieved when calibrating and testing with data from the same day is 81% (which occurs when the model trained on Day 1 data is calibrated and tested on Day 2 data). This indicates that the proposed technique achieves some level of channel portability.
Another result to note here is that the performance achieved by the proposed technique is on par with the ‘vanilla’ models. This can be seen by inspecting the accuracy of the ‘vanilla’ model trained and tested on Day 1 data in Fig. 6.3a and comparing this with the accuracy of the model trained with Day 2 data when it is calibrated and tested using Day 1 data in Fig. 6.3b. In this case, the model using the proposed technique outperforms the vanilla model by ∼12%. This excellent performance does not always occur though, as when the other available comparison is made (‘vanilla’ Day 2 model compared to Day 1 model calibrated and tested on Day 2) the model using the proposed technique under-performs the ‘vanilla’ model by ∼7%. Thus, by using the proposed technique for channel portability, accuracy reasonably close to a vanilla model trained on the target domain can be obtained.

6.3.3 Configuration Portability

To evaluate the configuration portability of the proposed technique, the configuration portability dataset listed in Table 5.1 was used to train four twin network models, each one using data exclusively involving one of the four transmitter configurations defined earlier in Table 5.2. Each of these trained models was then calibrated for each of the four configurations in the dataset. Finally, each possible calibrated model was tested with data from all four configurations.

Figs. 6.4a, 6.4b, 6.4c, and 6.4d show these results for models trained on data where the transmitters use Configs. 1, 2, 3, and 4 respectively. Note that all of these figures also show the results of training and testing a ‘vanilla’ model using
Figure 6.3: Channel Portability - Closed-set: Accuracy of models trained on data collected on (a) Day 1 and (b) Day 2, when tested on data from five different days. $N = 10\%$ of training data, $M = 10$ inputs per decision.
Figure 6.4: Configuration Portability - Closed-set: Accuracy of models trained on data collected from transmitters using (a) Config. 1, (b) Config. 2, (c) Config. 3, and (d) Config. 4 when tested on data from transmitters using four different configurations. $N = 10\%$ of training data, $M = 10$ inputs per decision.
Figure 6.5: Configuration Portability - Multiple Calibrations - Closed-set: Accuracy of models trained on data collected from transmitters using Config. 2 when calibrated with data from multiple configurations. $N = 10\%$ of training data (per configuration), $M = 10$ points per decision.

A first trend that can be seen in these figures is that models generally perform well when calibrated and tested on data from the same configuration. For instance, in Fig. 6.4a it can be seen that a model trained with data from transmitters using Config. 1 and calibrated with data from these same transmitters using Config. 2 performs well when tested on data from Config. 2. This trend holds for all tests performed in Fig. 6.4, and the average accuracy achieved when calibrating and testing on data from the same configuration is $\sim 71\%$. This trend indicates the ability of the proposed technique to be used to perform calibration to different transmitter configurations.
Another trend that is evident here, but is not present in the ‘hardware portability’ or ‘channel portability’ evaluations is that a model trained with data from transmitters using one of the configurations can be better calibrated to operate on data from more similar configurations. For instance, observing Fig. 6.4a, it can be seen that while reasonable performance is obtained when calibrating to any of the configurations, the best performance is obtained when calibrating to Config. 1 (the same configuration as training). Then, it can be seen that calibrating to Config. 2 produces slightly lower performance (∼77% accuracy), and calibrating to Configs. 3 and 4 produces even lower performance (∼50% accuracy). A similar trend exists across all sub-figures of Fig. 6.4, as the configuration with which the model was trained always performs the best, with accuracy dropping as the configuration the model is calibrated to is further from this original configuration.

The existence of this trend within this evaluation and its absence in the other closed-set evaluations can be explained by noting that changing the configurations of the transmitters is a more controlled change. That is, changing the hardware on which the data is captured produces a difference in the data that is based on the physical differences of the internal components of the receivers (which were not particularly selected to be related/unrelated), and changing the day on which the data is captured produces differences in the channel state that are not clearly controlled. In contrast, it is possible to set the transmitters to use a particular LoRa configuration, and furthermore, the LoRa configurations are clearly more and less similar to each other in terms of bit rate and spreading factor (see Table 5.2).

The final evaluation performed in this section leverages the notion that the
proposed technique could be used to calibrate for more than one configuration by performing the calibration algorithm multiple times. To this end, the twin network model trained using Config. 2 data was calibrated using data from multiple configurations and tested on data from all four configurations. The results of these tests are shown in Fig. 6.5. It can be seen in this figure that it is possible to calibrate a single model with data from all multiple configurations and that the resulting model performs reasonably well when tested on data from all of these configurations. It is worth noting, however, that there is a performance loss when compared with calibrating a trained model for a single configuration. For example, compare the performance of the model trained on Config. 2 data in Fig. 6.4b when it is calibrated and tested with Config. 1 data with the performance of this same model when calibrated for all four configurations and tested on Config. 1 data in Fig. 6.5. In this case, there is performance loss of \(\sim 20\%\).

6.4 Open-set Evaluation

In this section, models will be evaluated on the open-set, which includes both devices the models were trained on and other unknown devices. The goal in all cases is for the model to perform binary classification, deciding if an input originated from a known or unknown device.

All models were trained and tested using data from the same 10 wireless transmitters, where 75% of the data from a single transmission from each device was used for training, and 25% was reserved for testing (each transmission contains
156,250 examples of size 128x2). Test data from an additional 15 wireless transmitters was used to represent the unknown devices. In all tests, unless otherwise stated, the amount of calibration data used to perform the proposed technique ($N$ in Algorithm 1) was set to 10% of the size of the training data used to initially train the model and the number of input examples used to form the ‘input point’ ($M$ in Algorithm 2) was set to 10. It is also worth noting here that on all AUROC results in this section, a dashed line it plotted at 0.5 to indicate the performance of a random classifier.

Additionally, for comparison to the proposed technique, ‘vanilla’ models are also evaluated in this section. Note that the ‘vanilla’ models were only evaluated using the AUROC metric, as this metric does not require the choice of a threshold, while the TPR and FPR metrics do.

6.4.1 Hardware Portability

To evaluate hardware portability of the proposed technique on the open-set, the same twin network models used for closed-set evaluation of hardware portability were used. Recall that two models were trained for each of the six hardware portability datasets listed in Table 5.1: one using only the RX1 data, and one using only the RX2 data. Each of these models was then tested with data from both receivers under two conditions: when calibrated with data from the same receiver as training, and when calibrated with data from the receiver not used for training.
<table>
<thead>
<tr>
<th>Model</th>
<th>Indoor/RX1</th>
<th>Outdoor/RX1</th>
<th>Wired/RX1</th>
<th>Indoor/RX2</th>
<th>Outdoor/RX2</th>
<th>Wired/RX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla RX1 Cal. RX1 Cal. RX2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.6: Hardware Portability - Open-set**

Open set performance of models trained on data collected at (a)-(f) RX1 and (g)-(l) RX2 for each hardware portability dataset. $N = 10\%$ of training data, $M = 10$ points per decision.
Figs. 6.6a, 6.6b, 6.6c, 6.6d, 6.6e, and 6.6f show the results of these tests for the models trained using RX1 data, and Figs. 6.6g, 6.6h, 6.6i, 6.6j, 6.6k, and 6.6l show the results of these same tests for the models trained using RX2 data. In the sub-figures of Fig. 6.6 that report AUROC results, the results of training and testing ‘vanilla’ models on the same data are also shown for comparison.

First, considering the use of AUROC to measure open-set performance, some trends can be observed. The main trend is that in most cases, better AUROC is achieved when testing on the same receiver for which the model is calibrated. This can be seen in Fig. 6.6a, for instance, where the results when calibrating and testing on RX1 data (first three bars of RX1 calibration) and the results when calibrating and testing on RX2 data (final three bars) show higher AUROC when compared to the other tests. It is also notable that the ‘vanilla’ models (which use the max-logit decision score) perform far worse than the proposed technique in terms of AUROC when the models that use the proposed technique are calibrated and tested on the same receiver. This is indicative of the potential for the proposed technique to achieve hardware portability on the open-set.

Now considering the TPR metric, an even stronger case of the main trend that existed for the AUROC metric can be observed in Figs. 6.6b, 6.6e, 6.6h, 6.6k, where performance is strong when the model is calibrated and tested on the same receiver. This indicates that the proposed technique always achieves a high TPR, which means that is almost always correctly ’admits’ examples from known devices.

Finally, when considering the FPR metric, the trend is less clear. Ideally, models calibrated for a particular receiver would achieve a low FPR when tested
on data from that receiver, but this is not always the case. For example, in Figs. 6.6c and 6.6f, it can be seen that there are several cases where a model is calibrated for data from a receiver, but still achieves a higher FPR when tested on data from that receiver than when it is tested with data from a receiver for which it is not calibrated. For a particular example, notice the last two bars of the tests for models calibrated with RX2 data in Fig. 6.6c. These are both tests on RX2 data, so good performance would be expected, and thus low FPR. Instead these tests have the highest FPR of any test for models calibrated to RX2 in the figure.

This seemingly counter-intuitive result can be explained by carefully considering the meaning of FPR, and what exactly the model is attempting to do in the latent space during open-set evaluation. First, recall that when a model is calibrated using a particular set of data, for example, from receiver 1, that some data is provided to the model and it forms a centroid for each class in its latent output space. Now, achieving high TPR indicates that examples from the known classes lie very close to these centroids in the latent space. But, to achieve low FPR (which is desirable), the examples from unknown classes must form distinct ‘clusters’ in the latent space so that the model can distinguish known from unknown.

In the case where the model is calibrated and tested on data from the same domain (e.g. RX1), the ‘clusters’ from both the known and unknown devices will likely be closer to the centroids in the latent space than an alternative situation. This alternative is that the model is calibrated with data from one domain and tested on data from another (e.g. calibrated with RX1 data, tested on RX2 data). In this case, the ‘clusters’ in the test data from both known and unknown devices
will likely be farther from the centroids (which come from the calibration data) in the latent space. This means that when testing on a different domain than calibration, the model will more easily be able to ‘reject’ examples from unknown devices (achieving a lower FPR), but will also have a more difficult time ‘admitting’ examples from known devices (also achieving a lower TPR). Thus, achieving a low FPR is generally more difficult when testing on the same domain as calibration.

It is also worth noting that even when the FPR produced by calibrating and testing on data from the same receiver is not the highest FPR, the result is still not as low as would be desired. For example, observe Fig. 6.6c. Notice that the FPR for the model calibrated and tested on RX2 data from an ‘Indoor’ environment is among the lowest for all models calibrated with RX2 data, but that it is still in excess of 0.25. This is indicative of an issue with the Distances chosen during calibration and used for open-set decision making. It indicates that these Distances may be too large, producing a very higher TPR as well as a relatively high FPR.

6.4.2 Channel Portability

To evaluate channel portability of the proposed technique on the open-set, the same twin network models used for closed-set evaluation of channel portability were used. Recall that the channel portability dataset listed in Table 5.1 was used to train two twin network models: one using only data collected on Day 1 and one using only data collected on Day 2. Each of these trained models was then
calibrated for each of the five days in the dataset. Finally, each possible calibrated model was tested with data from all five days.

The results of this evaluation are shown for the model trained on data collected on Day 1 in Figs. 6.7a, 6.7b, and 6.7c, and for the model trained on data collected on Day 2 in Figs. 6.7d, 6.7e, and 6.7f. In the sub-figures of Fig. 6.7 that report AUROC results, the results of training and testing ‘vanilla’ models on the same data are also shown.

First considering the AUROC results in Figs. 6.7a and 6.7d, there are a couple of interesting trends. From observation of these figures it is evident that high AUROC is achieved when a model is calibrated with data collected on one day and tested using data from the same day. For instance, in Fig. 6.7a, the model calibrated with Day 2 data performs very well when tested with Day 2 data. This trend exists regardless of the calibration day, and for both the model trained on Day 1 data in Fig. 6.7a and the model trained on Day 2 data in Fig. 6.7d. The lowest Avg. AUROC achieved when models are calibrated and tested on data from the same day is $\sim 0.79$. This trend is indicative of the ability of the proposed technique to achieve portability to different wireless channels on the open-set.

Another trend of note from the AUROC results is that sometimes good performance is also achieved when testing using data from days other than the day used for calibration. For instance, in Fig. 6.7a, it can be seen that the model calibrated with Day 1 data also performs very well when tested with Day 2 data, and that the model calibrated with Day 4 data also performs very well when tested with Day 5 data. This trend is consistent, and is also present when the model is instead
trained with data from Day 2 in Fig. 6.7d. This could be indicative of some similarity between the wireless channel on these different days, but more work needs to be done to confirm this idea.

Moving to observe the TPR results in Figs. 6.7b and 6.7e, it can be seen that the desired trend is present. That is, TPR is high when a model is calibrated and tested on data from the same day. This indicates that the models using the proposed technique are excellent at correctly 'admitting' examples that come from known devices.

Turning to the FPR results, it can be seen that the trend present in the open-set 'hardware portability' evaluation is again present here. That is, the FPR results are less than ideal, since calibrating and testing with data from the same day does not always produce the lowest FPR. This can be attributed to the aforementioned idea that achieving a lower FPR is an easier task when the test data does not match the calibration data. It is also once again indicative of an issue with the calculation of the Distances during calibration and their use in the open-set decision algorithm.

6.4.3 Configuration Portability

To evaluate the configuration portability of the proposed technique on the open-set, the same twin network models used for closed-set evaluation of configuration portability were used. Recall that the configuration portability dataset in Table 5.1 was used to train four twin network models, each one using data exclusively from one of the four transmitter configurations defined earlier in Table 5.2. Each of these
Figure 6.7: Channel Portability - Open-set: Open set performance of models trained on data collected on (a)(b)(c) Day 1 and (d)(e)(f) Day 2, and tested on data from five different days. $N = 10\%$ of training data, $M = 10$ points per decision.
Figure 6.8: Configuration Portability - Open-set: Open-set performance of models trained on data collected from transmitters using (a)(b)(c) Config. 1, (d)(e)(f) Config. 2, (g)(h)(i) Config. 3, and (j)(k)(l) Config. 4, and tested on data from transmitters using four different configurations. $N = 10\%$ of training data, $M = 10$ points per decision.
Figure 6.9: Configuration Portability - Multiple Calibrations - Open-set: Open-set performance of models trained on data collected from transmitters using Config. 2 when calibrated with data from multiple configurations. \( N = 10\% \) of training data (per configuration), \( M = 10 \) points per decision.
trained models was then calibrated for each of the four configurations. Finally, each possible calibrated model was tested with data from all four configurations.

The results of this evaluation can be seen for a model trained on data from transmitters that use Config. 1 in Figs. 6.8a, 6.8b, and 6.8c. Also included in this figure are results for models trained using Configs. 2, 3, and 4 which are shown in Figs. 6.8d, 6.8e, and 6.8f, Figs. 6.8g, 6.8h, and 6.8i, and Figs. 6.8j, 6.8k, and 6.8l, respectively. It is also notable that the sub-figures of Fig. 6.8 that report AUROC results also show the results of training and testing ‘vanilla’ models on the same data.

First observing the results of evaluating using the AUROC metric in these figures, some interesting trends can be found. It can be seen that for the case of a model trained with any configuration, calibrating the model using Config. 1 or 2 results in the highest performance of the model occurring when it is tested with the same configuration as calibration (which is expected). One example of this can be seen in Fig. 6.8a, where the model calibrated with Config. 1 data performs the best when tested with Config. 1 data, and the model calibrated with Config. 2 data performs the best when tested with Config. 2 data. This trend is the same for all AUROC results in the figure, and is in line with the expected trends that have been mentioned thus far.

A unique trend present here can be seen by observing the performance of the models when calibrated and tested with Config. 3 data across all of the AUROC figures: Figs. 6.8a, 6.8d, 6.8g, and 6.8j. What can be seen is that when the model is trained with data from transmitters that use Config. 1 or Config. 2, the
performance of the model when calibrating with and then testing on Config. 3 data is not the best among all the tests for that calibration. In particular, testing with Config. 4 produces better performance than testing with Config. 3 when calibrating for Config. 3. This is not the case however, when the model is trained with data from transmitters that use Config. 3 or Config. 4. In this case, the result is as expected, where calibrating and testing with Config. 3 data produces the best result of all tests for that calibration. This result could be due to the relationships between the configurations mentioned in the closed-set evaluation section.

A final note on the AUROC results is that an anomaly not consistent with any other results occurred when training the model with data from transmitters that use Config. 3 and calibrating with data from Config. 4. This can be seen in Fig. 6.8g. In this case, none of the tests produced a reasonably good AUROC result. However, the relationship between the AUROC achieved by testing with each configuration when calibrating with Config. 4 is consistent with the other AUROC results. This poor performance could be due to the random selection of calibration data being poorly representative of the chosen class.

Despite the anomaly above, the TPR results for the models trained with all four configurations are consistent with the desired trend. That is, a very high TPR is achieved when the model is calibrated and testing using data from the same configuration.

In a similar way to the other open-set evaluations, the FPR results are less encouraging than those for AUROC and TPR. There are many instances where calibrating and testing using data from the same configuration results in the highest
FPR among all of the tests for that calibration. This could be explained using the same line of thinking from the ‘hardware portability’ open-set evaluation. That is, achieving a low FPR is easier when calibration and testing are done with data from two different configurations. This conclusion is actually reinforced by the FPR results here, as generally the FPR is lower when testing on a configuration that is ‘more different’ from the configuration used for calibration.

Finally, the last evaluation performed in this section leverages the notion that the proposed technique could be used to calibrate for more than one configuration by performing the calibration algorithm multiple times. To this end, the twin network model trained using Config. 2 data was calibrated using data from multiple configurations and tested on data from all four configurations. The results of these tests are shown in Fig. 6.9. The results here are encouraging for AUROC and TPR, but contain the same issues mentioned above when it comes to FPR results. In a similar fashion to the same test performed on the closed-set, there is also a notable drop in AUROC when calibrating for multiple configurations when compared to calibrating for a single configuration.

6.5 Effect of Number of Inputs per Decision

One of the interesting variables that exists in both Algorithm 2 and Algorithm 3 is the number of input examples, $M$, used to form the ‘input point’. This is the number of inputs the size of the network input (128x2) that are passed through the network and averaged together to make each decision.
The value of \( M \) was set to be 10 input examples for all of the evaluations performed thus far, but in this section, changing this value will be investigated. The results of this investigation are displayed for the ‘hardware portability’ scenario in Fig. 6.10a, for the ‘channel portability’ scenario in Fig. 6.10b, and for the ‘configuration portability’ scenario in Fig. 6.10c. Note that only a selection of all models trained were used for this investigation, and that only closed-set performance was measured. The results are also only displayed for calibrating and testing on the same domain, since altering the number of inputs used to make each decision did not affect performance in other cases.

Some trends that are present across all sub-figures of Fig. 6.10 are: (i) increasing \( M \) generally increases the accuracy of the models, and (ii) increasing \( M \) beyond 100 input examples cannot provide much improvement in model accuracy. The first trend makes some intuitive sense because increasing the value of \( M \) means that the model is making its decision based on more data from the input device. This additional data is used to form a point in the latent space for making a decision that will likely lie closer to the centroid for the correct class. In essence, using a larger value for \( M \) reduces the effect of input examples that are not perfectly characteristic of the input device.

It is notable that in almost all cases, the accuracy of the models is increased to almost 100% when the number of input examples in increased to 100. This may be an artifact of the fact that the same size test dataset was used regardless of the number of points used to make each decision. Thus, fewer decisions were made by the network for increasing values of \( M \). Note however, that even when \( M \)
was set to 100, the network made a total of about 3900 decisions during closed-set evaluation.

It is also worth reiterating here that typically it is not considered possible to perform this type of decision making based on multiple input examples in other applications of deep learning. For instance, in image classification tasks, typically one must make a single decision per input and cannot obtain more images from the same class to make a decision. However, there are applications within RF fingerprinting where it is reasonable to assume that many input examples from the same input device can be obtained. For instance, if the classification model was acting as a means of network authentication, obtaining multiple input examples before making a decision could simply require asking the input device to send more packets.

6.6 Effect of Amount of Calibration Data

Another interesting variable was introduced as a part of Algorithm 1, the calibration algorithm for the proposed technique. This variable is the value of \( N \), which is equal to the number of labeled examples used for calibration from each device. To perform all other evaluations, \( N \) was set to be equal to 10\% of training data used for each class (which also means 10\% of the total training data is used for calibration). This amounts to about 11,700 examples per class. In this section, we investigate the impact of changing the value of \( N \).

Since \( N \) is the number of points that will be used during calibration to calculate
Figure 6.10: Changing value of $M$: Effect of changing the value of $M$ in the decision making algorithm on accuracy of models from different evaluation scenarios: (a) hardware portability, (b) channel portability, and (c) configuration portability. The value of $N$ was fixed to 10% for all data shown.
a centroid that represents each class in the latent space, it intuitively makes sense to use a large number for $N$. However, since there are scenarios where it may not be possible to obtain many labeled examples from each class, the effect of using a reduced amount of calibration data was investigated. It is also of note that using a smaller value for $N$ decreases the amount of computation required to perform the calibration.

The results of this investigation are displayed for the ‘hardware portability’ scenario in Fig. 6.11a, for the ‘channel portability’ scenario in Fig. 6.11b, and for the ‘configuration portability’ scenario in Fig. 6.11c. It is notable that for each of these scenarios, only a selection of all of the models trained were evaluated to determine the effect of changing $N$. Additionally, only the results of calibrating and testing on the same domain are displayed, since altering the amount of calibration data did not significantly alter performance on domains the model was not calibrated for.

Some trends that are present across all sub-figures in Fig. 6.11 are: (i) increasing $N$ generally increases the accuracy of the models, (ii) increasing $N$ reduces uncertainty in the accuracy of the models (smaller confidence intervals), and (iii) there are diminishing returns for increasing $N$ beyond 0.1% of training data (this amounts to about 117 examples per class). These trends intuitively makes sense, as using more points to determine the Centroid (larger $N$) should produce a Centroid that is more representative of a desired domain, in turn producing a more accurate model. It also expected that using a very small value for $N$, such as 0.001% of training data (2 examples per class), causes the Centroid to be influenced too
heavily by each individual example, and thus the choice of examples has a great impact on the model accuracy (observe the wide confidence intervals for 0.0001%).

Interestingly, in Fig. 6.11a, it can be seen that the ‘Wired Same Tx’ model calibrated and tested on data from RX2 did not follow the general trends observed. The poor performance of this particular model was also observed earlier in this paper during both closed-set and open-set evaluation, and it is no surprise that it does not perform well here.

6.7 Computational Speed Considerations

It is of interest to assess how the proposed technique compares in terms of speed to traditional methods of adjusting to other domains, such as training a new DNN on data from a different receiver. For this, in lieu of attempting to measure the number of FLOPs (Floating Point Operations) or real time spent using the proposed technique vs. alternatives, a short explanation will illustrate how the calibration process is less processor intensive or quicker than alternatives that involve altering model parameters.

First, note that virtually all methods of altering the parameters of a DNN involve using some gradient descent process. This means that training data is required and gradients must be calculated. Thus, one could express the amount of time spent training or modifying the parameters of a DNN for one epoch ($T_1$) as the sum of the time spent doing forward passes (FP), the time spent doing loss calculation (LC), and the time spent doing gradient calculation, back-propagation
Figure 6.11: Changing value of $N$: Effect of changing the value of $N$ in the calibration algorithm on accuracy of models from different evaluation scenarios: (a) hardware portability, (b) channel portability, and (c) configuration portability. Note that $M$ was fixed to 10 points per decision for all data shown. (Error bars represent 95% confidence interval based on 5 trials with random calibration data).
and other updates (BP). That is, \( T_1 = FP_1 + LC + BP \).

Now, the time spent doing the calibration process \( (T_c) \) can be expressed as a sum of the number of forward passes required \( (FP_c) \), and some other distance calculations that do not involve gradients \( (OC) \). That is, \( T_c = FP_c + OC \).

One thing to note here is that in general, for the same size DNN and training data set, the inequality \( FP_c \leq \frac{1}{10}FP_1 \) must hold. This is due to the fact that the number of forward passes required to train a network for one epoch \( (FP_1) \) is equal to the number of training examples (i.e. all examples must have been passed forward through the network before back-propagation can happen), while the number of forward passes required for the calibration process is only a fraction of the number of training examples (10% here).

Additionally, it can be posited that \( OC \leq LC + BP \), because the \( OC \) term does not involve gradient calculation, which means that in general \( T_c \leq T_1 \).

Considering that multiple epochs are typically required to train a DNN, even if it has been pre-trained, it is safe to say that the proposed calibration method is quicker than any alternative that involves training a DNN by many orders of magnitude. For instance, for the evaluations performed in this work, training the DNNs took hours, but performing calibration took minutes only.
Chapter 7: Open Research Challenges & Directions

The following are several key challenges and future directions that have been made evident through this work.

- **Threshold value selection.** The first of these challenges relates to the open-set performance of the proposed technique. From the results, it is clear that the open-set performance of the technique (in terms of TPR and FPR) is less than desirable. That is, the balance between the TPR and FPR is not properly tuned, as the FPR is too high (resulting in many unknown devices being classified as known). This indicates that the per-device Distances calculated during the calibration process (which are used as thresholds for open-set decision making) may be too large. Future works should seek to devise a better method for determining these Distances, while still maintaining the computational advantage of a quick calibration process. Some quick way of performing higher-order statistical analysis on the calibration data in the latent space could be conceived.

- **Latent space dimensions.** There are also additional variables that could be explored to produce better performance with the proposed technique. These include the dimension of the latent space at the output of the neural network, which could be increased to perhaps encourage clusters to form
farther apart in the latent space. The structure of the neural network itself is also a variable that could be worth exploring. In particular, training a deeper network might provide better performance.

- **Portability scalability.** Future work should also seek to expand upon this work by testing the proposed technique with increased scale. This would mean first and foremost testing classification models with an increased number of transmitters and using an increased number of receivers to test hardware portability. Additionally, testing hardware portability to different receiver models would be a good indicator of the real-world applicability of the proposed technique.

- **Real-timeness consideration.** Another future direction would be to perform real-time experimentation to better quantify the amount of time that the calibration process takes and thus better emphasize its advantage over other more processor-intensive methods. In particular, experimentation using IOT devices to perform the calibration process could be useful.

- **Transmitter hardware portability.** A final possibility for future work could be testing the portability of the model with respect to the transmitters themselves. That is, calibrating with labeled data from transmitters that the model did not train on. If this concept worked, it would enable an end user to train a model using one set of devices, and then simply collect several samples from new devices to perform classification on these new devices. This idea has shown some promise as a proof of concept with limited evaluation.
Chapter 8: Conclusion

In conclusion, the technique proposed in this work posits that a portable deep neural network for RF fingerprinting can be created which can be calibrated quickly, using a small amount of labeled examples, to perform effectively on data from a domain other than the one it was originally trained with. It has been demonstrated through experimentation with a testbed of RF devices that this is possible for closed-set classification when the change in domain is due to: a change in the receiver hardware, a change in the wireless channel due to the passage of time, or a change in the configuration of the transmitters. It has also been demonstrated through experimentation that this same result may be possible to achieve when performing open-set classification, albeit tuning of the proposed technique will be required to achieve better performance on the open-set.

To sum up: the proposed technique achieves its objectives of portability with minimal labeled data and computational restrictions, giving it an advantage over other techniques for portability. With additional tuning and research, this technique could be applied in various real-world RF fingerprinting applications.
Bibliography


