HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Department: Information and communication technologies
Faculty: Information technology and communication systems
Orientation: Software engineering

Bachelor Thesis

RF fingerprinting on NFC devices

Not confidential

Student: Luc Wachter

Project proposed by: Joël Conus
Kudelski Group SA
22-24, Route de Genève
1033 Cheseaux-sur-Lausanne

Teacher in charge: Alberto Dassatti

Academic year: 2019-2020

Combremont-le-Grand, 31st July 2020

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Information and communication technologies
Information technology and communication systems
Software engineering
Student: Luc Wachter
Teacher in charge: Alberto Dassatti

Bachelor thesis 2019-2020

RF fingerprinting on NFC devices

Company name

Kudelski Group SA

Publishable summary

NFC, Bluetooth, Wi-Fi and every common wireless communication protocol operate using radio waves. Emitting in the Radio Frequency (RF) spectrum requires complex analog components that, because of their nature, cannot be completely identical to one another. The inherent imperfections of these analog components are what makes RF fingerprinting possible. Indeed, this technique theoretically allows the identification of a device just through the analysis of its signal.

This project's goal is to determine whether applying machine learning techniques to the problem of RF fingerprinting is effective for the NFC protocol, which is known to be vulnerable to relay attacks, for example. To do this, we first needed to build a dataset of NFC communications using Software Defined Radio (SDR) equipment. Then, we had to apply signal processing algorithms to process the data and finally, we built supervised deep learning systems in order to classify the tags.

The project was very much an exploratory endeavour, so a lot of attention was given to the state of the art. Existing research in the field of RF Machine Learning (RFML) guided our experiments. After this analysis phase, a lot of work went into configuring and testing the SDR setup to acquire the data. We then wrote the software to manipulate this data and format it in order to feed it to our learning algorithms.

In terms of results, our model can classify a signal segment from a known device up to around 85% of the time, depending on the number of tags. The system is quite limited, and not yet scalable in any meaningful way, but it does answer some questions and open paths for future developments.

1 Preamble

This Bachelor thesis (hereafter BT) is conducted at the end of the curriculum, with the goal of obtaining the title of Bachelor of Science HES-SO in Engineering.

As an academic project, its content, without assuming its value, engages neither the author's responsibility, nor these of the jury of the Bachelor thesis and the School.

Any use, even partial, of this BT must be done with due regard to copyright.

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Vincent Peiris
Chef de département TIC

Yverdon-les-Bains, 31st July 2020

2 Authentication

The undersigned, Luc Wachter, hereby certifies that he has carried out this work and has not used any other source than those expressly mentioned.

Le soussigné, Luc Wachter, atteste par la présente avoir réalisé ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Combremont-le-Grand, 31st July 2020

Luc Wachter

3 Initial project description

Radio Frequency (RF) fingerprinting is a technique that allows the identification of radio transmitters (such as Internet of Things (IoT) devices) by analysing the spectrum of their transmissions. Indeed a device's spectrum is unique because of tiny imperfections in the manufacturing process of its analog components. Analysing a device's spectrum can typically be done using machine learning algorithms.

Near-Field Communication (NFC) technology is often used in access control and payment applications but many implementations are vulnerable to relay attacks. This type of attacks allows an attacker to relay messages between a reader and an NFC device without the knowledge of the device's owner. Doing this effectively convinces the reader it is communicating with the legitimate device. Research and tools that facilitate such attacks are publicly available.

The goal of this project is to determine if RF fingerprinting could be used as an authentication technique against relay attacks.

The main steps of this project are the following:

- Build a simple lab setup with Software-Defined Radio (SDR) equipment to acquire signals between an NFC device and its reader
- Acquire RF spectrum data of various NFC devices
- Analyse the signals
- Classify the signals of the devices by using supervised machine learning classification techniques in order to differentiate trusted devices from attacker / relay devices
- Determine if this identification technique could be used as an authentication feature against relay attacks

As the receiving equipment (SDR) has an influence on the recorded signals, for this project we consider a single receiver to record the RF samples. Similarly, the lab setup should be built to provide an ideal low-noise and low-interference environment to simplify the analysis phase.

The expected deliverables are the following:

- A tool able to identify NFC devices by analysing the RF spectrum of their signals, at least in an ideal environment and with a small number of devices
- A detailed account of the steps taken and the setup used (as part of the report)
- An analysis of the results (as part of the report)

Collaboration with other researchers in this field is wished (EPFL, ElectroSense).

Table of contents

List of Figures

List of Tables

List of acronyms

ADC Analog-to-Digital Converter.

ASK Amplitude Shift Keying.

BLE Bluetooth Low Energy.

CNN Convolutional Neural Network.

DAC Digital-to-Analog Converter.

DNN Deep Neural Network.

GRC GNU Radio Companion.

HF High Frequency.

I/Q Signal composed of In-phase and Quadrature components.

MLP Multi-Layer Perceptron.

NFC Near-Field Communication.

OOK On-Off Keying.

PCD Proximity Coupling Device.

PHY Physical layer.

PICC Proximity Inductive Coupling Card.

PUF Physical Unclonable Function.

RF Radio Frequency.

RFML Radio Frequency Machine Learning.

SDR Software-Defined Radio.

4 Introduction

4.1 Project description

Small imperfections in the electromagnetic emissions of radio transmitters make it possible to identify them based only on the way they transmit. This is called Radio Frequency (RF) fingerprinting and it is possible thanks to tiny manufacturing imperfections in the devices' analog components. Using Software-Defined Radio (SDR) equipment, we can analyse this spectrum in order to extract the aforementioned differences and identify a device.

Such techniques can be used on any type of radio transmission: Bluetooth, Bluetooth Low Energy (BLE), WiFi, LTE (part of 4G mobile networks), etc. This project aims to use RF fingerprinting on NFC devices. Indeed, NFC is often used in access control and payment applications but many implementations are vulnerable to relay attacks. Spoofing the imperfections in an emitter's radio spectrum is close to impossible at the present time, since it is essentially a hardware signature. This is why a technique like the one described here would be a valuable additional security layer.

The goal of this project is to determine whether applying machine learning techniques to the problem of RF fingerprinting NFC devices could be used as an authentication technique, in order to prevent relay attacks. The first step to achieve this goal is to produce a dataset of raw NFC transmissions using SDR. Indeed, to our knowledge, there exists no available dataset of raw NFC captures.

4.2 Context

This project is conducted in the context of a bachelor thesis at the School of Engineering and Management (HEIG-VD), the largest branch of the University of Applied Sciences - Western Switzerland (HES-SO).

- Department: Information and communication technologies
- Faculty: Information technology and communication systems
- Orientation: Software engineering

It was proposed by Mr Joël Conus of Kudelski Group SA.

4.3 Document description

As this project is largely of exploratory nature, the report is of prime importance. It highlights the steps taken during the project, the experiments and the results. It will be structured as follows.

We will first introduce important concepts for the understanding of this document. Then, we will study previous works in the field in order to better define our problem and to discern the obstacles we might face. After this analysis phase, we will tackle the first practical part of the project: designing an acquisition setup for the data and building our dataset. Finally, said dataset will be used to train machine learning models, whose performance we will discuss in the last part of this document.

The source code for this project, as well as the datasets built for it are available on this repository: https://github.com/Laykel/nfc-rf-fingerprinting.

5 Necessary concepts

Before analysing the problem and the existing works on the topic and before describing the hardware used and the data captured, it is worth explaining some of the concepts behind SDR and NFC. We will only discuss the elements needed to understand this document, and will refer the reader to other works for details. Readers who are acquainted to software-defined radio and NFC's characteristics can skip this section and continue at section 6.

5.1 Software-defined radio

First, we will note that radio waves are electromagnetic radiations operating in the Radio Frequency (RF) range. RF is a portion of the electromagnetic spectrum generally comprised between ∼3kHz and 300GHz [1]. It is used for radio communications of all sorts.

The main principle behind software-defined radio, as its name suggests, is to make as many of the elements of a traditional radio's pipeline digital. Doing this makes it possible to use a computer's processor to do the signal processing tasks that once required specialized hardware. Of course, it is not functional to simply strap an antenna to an Analog-to-Digital Converter (ADC) and do everything else in software. We still need some analog components in front of the ADC to preprocess the signal and ensure a consistent sampling. [2, 3, 4]

These analog components are combined in SDR hardware available to buy. Popular examples include the cheap RTL-SDR dongle[1] (receiver) and the HackRF[2] (transceiver). Once an SDR is plugged into a computer, the proper drivers and software are installed and an antenna is connected to the device, anybody is able to receive and process a wide range of frequencies. This range is limited by the components in the SDR hardware and by the antenna.

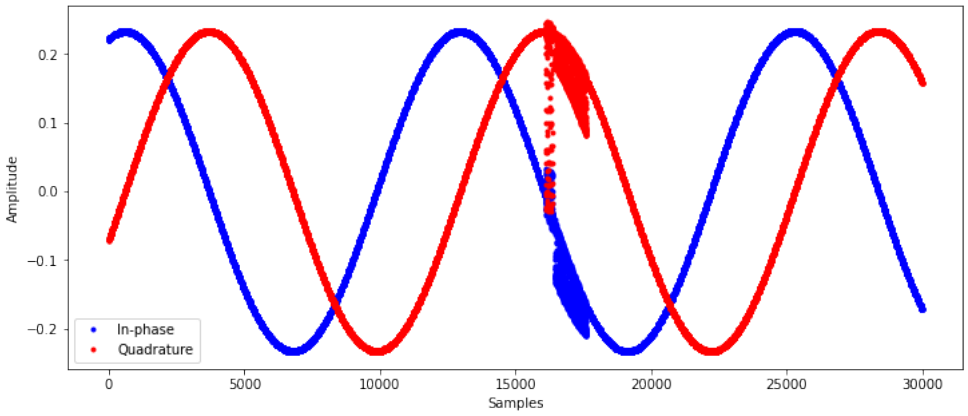

Figure 1: In-phase and quadrature components of a signal

To finish this introduction to SDR, we will describe the representation of the data. Figure 1 shows a graph of the samples returned by the SDR pipeline as dots. As can be seen, the signal is represented using two components. The "quadrature" component is phase shifted by 90 degrees in relation to the "in-phase" component. Every sample can be represented by a complex number where the real part adds to the in-phase component and the imaginary part adds to the quadrature component. Such a representation allows us to get a lot more information from the signal. [5, 6]

[1] https://www.rtl-sdr.com/about-rtl-sdr
[2] https://greatscottgadgets.com/hackrf/one

Here, the X axis represents the sample number (starting at 0) rather than a time value. The link between the sample number and the time elapsed is the sampling rate (F_s), expressed in number of samples per second (S/s). For example, considering a sampling rate of 3MS/s, figure 1 shows a signal over 10ms: $\frac{30000[S]}{3000000[S/s]} = 0.01[s]$.

In this document, we will often use the magnitudes representation of a signal. Said representation is built by taking the magnitude (or module) of each complex sample (calculated with $\sqrt{I^2 + Q^2}$) and plotting it. Figure 2 shows the magnitudes of the samples in figure 1.

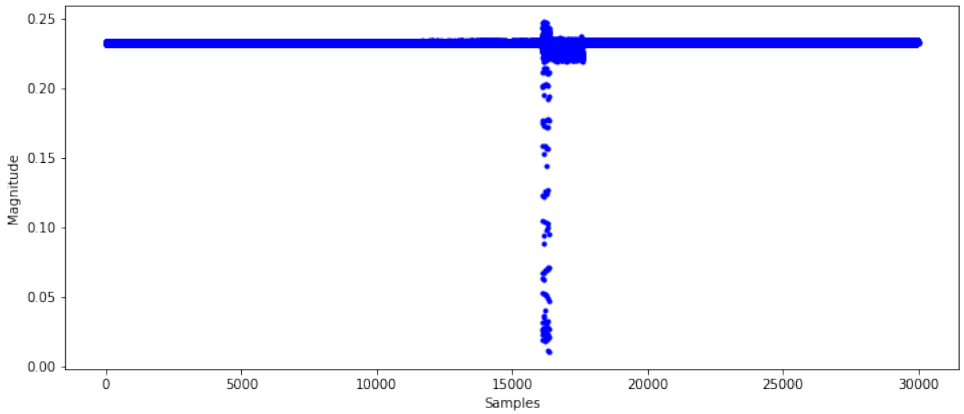

Figure 2: A signal represented as the magnitudes of its samples

It is important to note that neither representation shown here use absolute units. The amplitude reported in these figures is the one returned by our SDR setup, which is not a calibrated instrument.

5.2 Near-field communication

In order for our radio system to be effective, it needs to be adapted to the protocols used. This is why we need to understand how NFC operates.

NFC is the name for a group of communication protocols designed for small distance (max. 10cm) transactions. The idea is that a reader can supply power to a passive tag and get the information stored in it. The reader devices are sometimes called initiators or PCD (for Proximity Coupling Device) and the tags are sometimes called targets or PICC (for Proximity Inductive Coupling Card).

Operating at a frequency of 13.56MHz, NFC is well in the High Frequency (HF) range of 3 to 30MHz. This is substantially lower than most communication protocols like WiFi (2.4GHz or 5GHz), Bluetooth (2.4GHz) or ZigBee (868MHz, 915MHz or 2.4GHz).

Also in contrast to these other protocols, NFC's modulation scheme is On-Off Keying (OOK) which is a form of Amplitude Shift Keying (ASK). (Except for NFC type B, which uses Binary Phase Shift Keying (BPSK) in target to initiator mode.)

Because it is designed to work only in close proximity, NFC uses inductive coupling between devices to transmit information. In simple terms, this means the reader can only send a signal as far as its generated magnetic field goes. The passive tag responds by modulating the same

magnetic field. It also means that far-field interferences from radio devices operating in the same frequency range practically don't affect an NFC communication. [7]

If we capture a typical NFC communication using SDR hardware and represent it as magnitudes, we'll see something like figure 3. We can observe that before the reader is brought to the tag, the line is almost flat at zero. This shows a very low noise level. Then, there is a sudden surge of amplitude in the signal as the reader generates the magnetic field required for the communication. This part, referred to as the "transient portion" of the signal by Xu et al. [8] (although they talk about WiFi), might be very characteristic of the device. Then, the amplitude decreases and regular request/responses start.

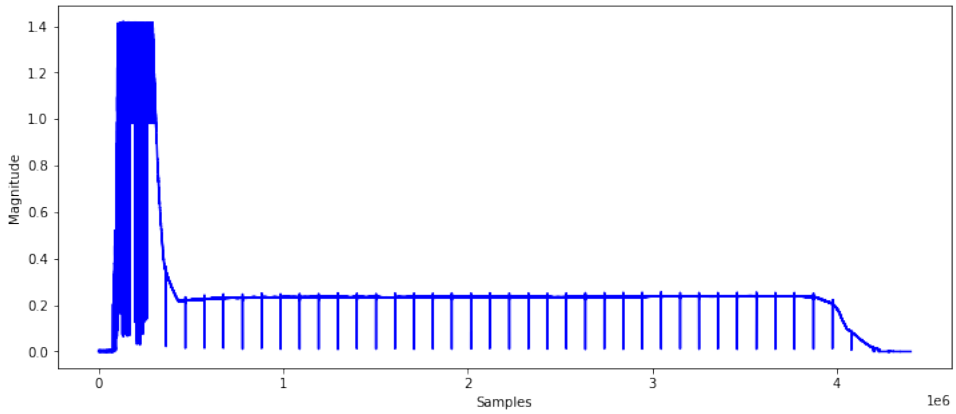

Figure 3: NFC communication represented as magnitudes

In figure 4, we take a closer look at one of the request/responses of the previous figure. We can clearly see a first transmission followed by a pause and then a second transmission, with smaller amplitude. This is how NFC communications work, with a request sent by the reader device followed by the tag's response through modulation of the reader's magnetic field.

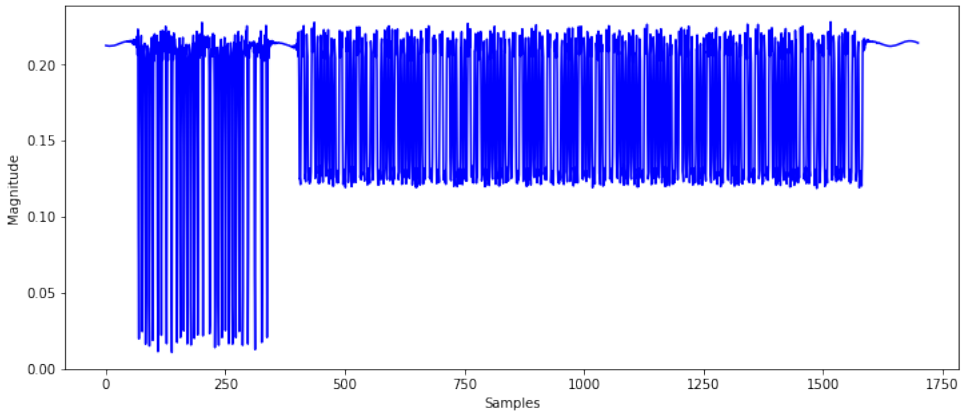

Figure 4: NFC single request/response represented as magnitudes

6 State of the art

6.1 Taxonomy

It is certainly useful to start with a review of the different ways to categorize the features and algorithms used by researchers in the field of radio frequency fingerprinting. More specifically, we'll focus on Radio Frequency Machine Learning (RFML) research, though other fingerprinting techniques exist that don't involve machine learning. The goal is to define our needs precisely and select the important factors to consider.

6.1.1 Taxonomy for features

The features we select must allow us to identify a precise device among potentially very similar devices. We need what Delgado et al. [9] describe as a Physical Unclonable Function (PUF). PUFs are physical distortions that are unique to a specific system. They are another way of talking about fingerprints.

Xu et al. [8] propose three ways to categorize radio signal features:

- based on the specificity of the feature (from vendor specific to device specific),
- based on the layers (PHY, MAC, Network and higher),
- and based on the acquisition method (passive or active).

Features from the MAC and higher layers typically require in depth knowledge of the protocols in play. Not only that, but they also tend to be less specific than we would like (either vendor specific or depending on the type of device). This indicates we should probably focus on the physical (PHY) layer features, which rely on imperfections in the manufacturing process of the devices.

6.1.2 Taxonomy for fingerprinting algorithms

Riyaz et al. [10] provide a visual categorization of fingerprinting approaches, which we adapt in figure 5. We take a look at these approaches in the following paragraphs.

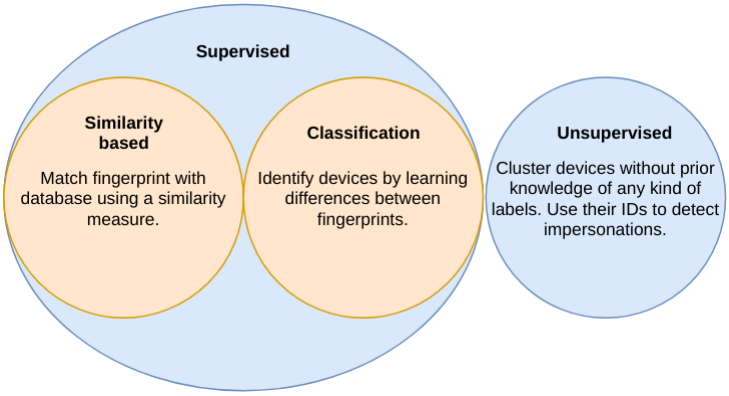

Figure 5: Fingerprinting algorithms taxonomy

Supervised approaches: Supervised approaches use features from labelled data to generate a function capable of separating the different classes. These approaches can be further categorized in similarity based and classification techniques.

Similarity based techniques are white-list algorithm that use a database of fingerprints and a similarity measure to determine whether a device is legitimate. Developing a technique like this usually requires prior knowledge of vendor specific device features [10].

Classification systems can also require deep knowledge of the features and protocols used, in the case of "traditional", manually tuned classifiers. Those are built to extract predetermined features and work similarly to other white-list algorithms afterwards.

In this age of deep learning though, research seems to be more interested in classification techniques that are able to extract the features they need by themselves. This can be done through deep Multi-Layer Perceptrons (MLP) [9, 11] or through more advanced techniques like Convolutional Neural Networks (CNN) [10, 12, 13, 14, 15]. The latter have proved very powerful in domains like computer vision, natural language processing and recommendation systems. This success is one of the reasons experimentations on CNNs are common in RFML research.

Unsupervised systems: Unsupervised approaches cannot by themselves discriminate a legitimate device from an illegitimate one. They don't have that information, since they work with unlabeled data. In order to detect attempts of impersonation, such a system has to keep a record of fingerprints and linked identifiers (MAC addresses, serial numbers...). It can then throw an alert and update a black-list when multiple fingerprints are linked to the same ID or when multiple IDs are linked to the same fingerprint [8, 16].

Xu et al. [8] specify that unsupervised approaches are appropriate when the fingerprints of legitimate devices are not available.

6.2 Acquisition

The vast majority of the research considered for this project uses USRP systems to record transmissions as raw I/Q signals. The number of devices can be anywhere from 5 to 500 (but most often less than 20). They all use data acquired from WiFi (802.11) or Zigbee (802.15.4) devices. [10, 12, 13, 14, 15, 16]

Some, like Sankhe et al. [15], also describe how they add artificially induced impairments to simulated signals with MatLab. This is something we could also do, but it is secondary. The focus is on the creation of a robust dataset of real world transmissions between an NFC reader and tags.

Although it isn't specified as such in any of the considered articles, we can deduce that in order for the dataset to be robust, it needs to answer some basic criteria. The following list assumes the dataset will be used to train a machine learning model to discriminate between passive tags.

- It should contain recordings of both very similar and very different devices.
- The data transmitted should not be a discriminating factor.
- The number of devices should be high enough to analyse the scalability of the solution.
- Only one reader should be used to initiate a communication.

- Only one SDR should be used to capture the data.
- The capture's parameters should be constant across captures.
- At the same time, some variability in the signal's amplitude and phase may help the solution to be more general.

6.3 Features

6.3.1 Features selection

Even if we don't plan to manually select and extract the features that will form the fingerprints of our devices, it is useful to learn about them. It will allow us to know whether they can be found in a given dataset or not, what to look for in theory, and what preprocessing methods could be useful to magnify them.

Whether we end up with a system that is able to identify many devices uniquely, or one that only tries to separate a specific device from the others, we will need device specific accuracy. We don't want to make relay attacks impossible only if the attacker doesn't use a device from the same vendor as the victim's device. This is why in section 6.1.1, we concluded that the features we are most interested in are from the physical layer.

Because of their nature, these features should be appropriate no matter the protocol used. The following list is composed of features described by Riyaz et al. [10] and also used in other works.

- I/Q imbalance: The amplitude and the phase are not exactly the same on the in-phase and the quadrature signals, because of the imperfections in the quadrature mixers.
- Phase noise: When the baseband signal is up-converted to the carrier frequency, it is sensible to phase noise which creates rotational vibrations.
- Carrier frequency offset: The difference between the carrier frequency of the transmitter and the carrier frequency of the receiver.

We chose to mention these features because of what they can teach us about the task at hand. They meet the criteria we previously established and are an example of what we would like our end models to extract. We won't extract them manually but learning about them is still important to understand the subject.

6.3.2 Preprocessing

It is clear that preprocessing the data appropriately can greatly increase the accuracy of a model and reduce its complexity.

The first question to ask is how should the data be partitioned, in order to be fed to the learning algorithm. This question will be discussed in section 6.4.4 since it is identical to choosing the input of our model.

Several works mention wavelet transforms (either discrete or continuous) as effective means to amplify the characteristic features of a wireless device [8, 12, 13]. They report increased accuracy and scalability, and reduced complexity. It is certainly worth it to explore this preprocessing method.

6.4 Machine learning applied to radio frequency

6.4.1 Challenges

Riyaz et al. [10] highlight some of the challenges faced when working on RFML problems. They are reformulated in the three first items of the list below. The next items are personal additions.

1. Finding the optimal partition length to feed the learning algorithm.
2. Finding the optimal network architecture for the problem (in the case of neural networks).
3. The absence of standard datasets to train and evaluate a model.
4. Finding a cheap preprocessing method that improves performance and reduces complexity.
5. Achieving a demonstrably scalable system.

Great insight into item 1 is given by Youssef et al. [13]. Indeed, they compare the performance of models trained with different input segment sizes. We delve deeper into this matter as well as item 2 in section 6.4.4.

Item 3 is the core matter of the first part of our project. Indeed a considerable portion of our work consists in the elaboration of a dataset of NFC transmissions for different PICC devices.

6.4.2 Supervised or unsupervised

The aim of this project is to explore supervised learning techniques. This is why most of the literature considered here studies supervised systems. Also, in general, it seems works that use supervised methods are a lot more abundant than their unsupervised counterparts. This could be because of the popularity of models such as convolutional neural networks, and their apparent adaptability to the problem of RF fingerprinting.

Despite this, we can see that unsupervised learning techniques are also showing promising results. An example of this is the work of Nguyen et al. [16]. The paragraph about unsupervised systems in section 6.1.2 describes the basics of their system pretty well. They use a Nonparametric Bayesian model to detect the number of devices and then cluster them based on their fingerprints. This technique allows them to discriminate between an unknown number of devices that the model never encountered before. They show good results with four devices using two features strictly from the PHY layer.

While this work is interesting, it uses pre-engineered features. We were unable to find research on deep unsupervised learning techniques (such as deep belief networks) for the problem. Such a solution could be an interesting topic for another project.

6.4.3 Comparing supervised approaches

Several articles have compared the performance of different machine learning approaches. In the next paragraphs we discuss the findings of two of them.

Riyaz et al. [10] compared their Convolutional Neural Network (CNN) with the techniques of Support Vector Machine (SVM) and logistic regression. They report substantially higher performance using their CNN (up to 60-70% better accuracy). Their results, although limited in the number of devices (they only classify up to five), also seem to show that CNNs are more

scalable than the other methods. Indeed, increasing the number of classes caused a significant drop in the performance of the other algorithms, but not for the CNN.

Youssef et al. [13] compare the performance of four supervised algorithms: SVM, Deep Neural Networks, CNN and Accelerated Levenberg-Marquardt Multi-Stage Training (A-LM MST). The latter is an advanced classification technique that makes training deep neural networks less resource heavy by using a hierarchy of smaller Multi-Layer Perceptrons (MLP) rather than one big MLP. They use data collected from 12 WiFi transmitters.

Their results show impressive performance for their MST, especially when provided with very little data. Their CNN model is close second, although it performed significantly worse with very little data. Then comes the DNN with pretty good results and then only the SVM.

These results can guide us in our choices of experimentation. They show that SVM algorithms, while capable to some extent, are not appropriate for this specific problem. They also show CNNs are very promising, more scalable and more adaptable than "simple" DNNs. The MST solution is very interesting, but reducing the computational complexity is not our primary concern and the improvement doesn't seem that substantial.

6.4.4 Neural network architectures

In section 6.4.1, we noted that one of the challenges we face is to find the optimal partition length for the input of our model. To give an element of answer, Youssef et al. [13] don't only compare the performance of 4 different models, they also do it with five different partition sizes (32, 64, 128, 256 and 512). They find their DNN architecture handles short segments better than long ones, while their CNN gets progressively more performant as the segments get larger.

Stankowicz et al. [11] compare different kinds of formats for their inputs: two real valued series (one for the I component and one for the Q component), one complex valued series and one complex valued series with a spectrogram added as additional information. Their results are not the clearest, but they seem to show better performance when using a single series of complex values as input. They also show better performance in general for models that use complex valued weights. This is an interesting argument that we could explore.

Another of the challenges previously noted is finding the optimal architecture for a neural network in the context of RF fingerprinting. Studying the various architectures employed in previous works can help us define what is effective and what direction to take for our own experiences. We won't describe specific architectures in this section, but will reference elements from them when conceiving our experiments.

In general, the works cited throughout this section train their models to output the ID of a specific device. In this project, we would like to try and do the same for NFC tags. Elements like incremental learning (the ability to train the model again with more devices without starting from zero) and shift invariance (the ability not to care about the position of values in the input) are also important to take into consideration.

7 Dataset creation

The first step of any classification project, especially if it uses machine learning, is to acquire and refine data. At the time the project was proposed, it was already clear that no existing dataset would be available. This proved to be true as discussed in section 6, where we saw that existing research focuses on WiFi and Zigbee technologies.

7.1 Radio setup

This section's goal is to describe the material used to capture the communications between NFC readers and tags. The final setup is illustrated in figure 6.

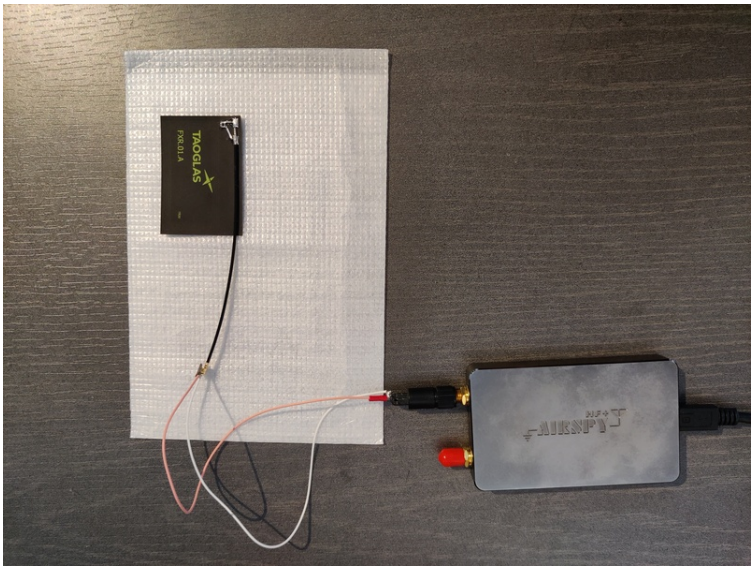

Figure 6: Airspy HF+ and antenna setup

7.1.1 The SDR

The first part of the analysis phase was conducted using a LimeSDR Mini[3]. We used it to learn about SDR in general and to make our first recordings. In that regard it was very useful, but our model had a set of shortcomings we weren't able to accommodate. First, it became very hot very quickly, which couldn't have been good for the stability of the recording. Also, while it worked all the time for higher frequencies, it seemed to only pick up our HF signals one out of six times or so. This proved quite frustrating and attempts at tweaking the parameters in LimeSuite GUI (the device's official configuration software) generally resulted in errors.

Despite these setbacks, we were able to record communications well enough to decode the reader's transmissions, as described in section 7.6. The tag's response, though, was drowned in the noise most of the time, as far as we can tell. These are the reasons why we replaced the LimeSDR Mini with the Airspy HF+[4] you can see in figure 6, courtesy of Mr Joël Conus.

The Airspy HF+, as its name suggests, is built for HF (the frequency range in which NFC operates). As such, it proved a lot more stable at 13.56MHz (picking up our signal every time)

[3] https://www.crowdsupply.com/lime-micro/limesdr-mini
[4] https://airspy.com/airspy-hf-plus

and a lot less prone to heating. Most importantly, the noise level is much lower with this device, which allows us to clearly distinguish a tag's response. The only drawback is its sampling rate which can only be set at one of seven values, the highest of which is 912kS/s. (There used to be only five possible values, before we applied the latest firmware patch for the device.) [17, 18]

Name	Frequency range	Bandwidth	Transmit
LimeSDR Mini	10MHz - 3.5GHz	Up to 30.72MHz	Yes
Airspy HF+	HF: 9kHz - 31MHz VHF: 60MHz - 260MHz	912kHz, 768kHz, 456kHz, 384kHz, 192kHz, 96kHz or 48kHz	No

Table 1: Theoretical characteristics of mentioned SDRs

7.1.2 The antenna

As described in section 5.2, NFC uses inductive coupling rather than the more common far-field electromagnetic radiations. Because of this, our system needs a near-field antenna, which in this case is really just an inductor. A simple loop of copper wire qualifies as such, but in order for our system to be perfectly tuned to 13.56MHz, we used an industrial antenna: the Taoglas FXR.01.A[5].

The antenna should be placed at least 15mm away from metallic objects, for they interfere with the magnetic field used for the communication.

As can be seen in figure 6, the adapter between the SDR and the antenna is homemade using spare connectors and copper wires. The risk of interferences because of this rather unsophisticated adapter is noted, but doesn't seem to be significant later in the work.

[5] https://www.taoglas.com/product/fxr01-nfc-flex-reader-antenna

7.2 Software used

7.2.1 Acquisition

We used GNU Radio Companion (GRC)[6] for all of our data captures. It is a very versatile tool, allowing us to define software pipelines using a block interface to create flow graphs. As it compiles to python, the idea is to use it as a base for acquisition and processing scripts.

Figure 7 shows a simple flow graph used to read a previously recorded signal and to experiment with parameters like a low-pass filter.

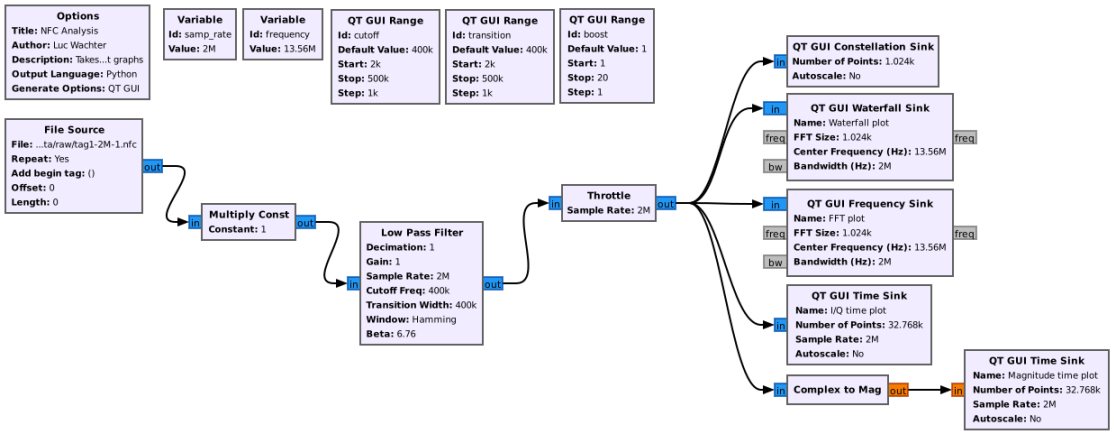

Figure 7: Example of a GNU Radio Companion flow graph

The format used by GRC's file writer is simple. It writes raw bytes into a file, alternating between the real part and the imaginary part of the sample. Both parts are written as 32 bits floating-point numbers.

7.2.2 Tag manipulation

On the reader (an Android smartphone), we used the NFC Tools[7] application to get information about the tags and manipulate their content. The tool makes use of Android's NFC capacity and of the smartphone's hardware to communicate with passive tags.

7.3 Inventory of devices

Here, we list the devices used to create the dataset. These include the PCDs (readers) and the PICCs (tags) whose communications were captured.

In terms of readers, table 2 lists the few devices used, for documentation purposes. Of course, only one reader will be used to elaborate the final dataset, but it was useful to compare the results during the analysis phase. We weren't able to find a specific NFC chip in either smartphone's characteristics. The final dataset will be created with the help of `reader1`, as it is more recent.

On the other hand, the list of tags and their technical details can be found in table 3. A picture of tags 1 to 7 is also provided in figure 8. As the table shows, tags 1 to 5 use the exact same

[6] https://wiki.gnuradio.org/index.php/Main_Page
[7] https://www.wakdev.com/en/apps/nfc-tools.html

Name	Type	Model
reader1	Smartphone	OnePlus 8
reader2	Smartphone	Nokia 7+

Table 2: Inventory of PCD devices

chip model. Tags 1 to 8 are all NFC type A compliant, while tag 9 uses the FeliCa standard from Sony. It will be interesting to contrast the classification performance between tags of the same type and between tags of different types.

Name	NFC type	Standard	Chip	Writable	Storage	Bit rate
tag1	NFC-A	ISO 14443-3A	NTAG213	Yes	137B	106kb/s
tag2	NFC-A	ISO 14443-3A	NTAG213	Yes	137B	106kb/s
tag3	NFC-A	ISO 14443-3A	NTAG213	Yes	137B	106kb/s
tag4	NFC-A	ISO 14443-3A	NTAG213	Yes	137B	106kb/s
tag5	NFC-A	ISO 14443-3A	NTAG213	Yes	137B	106kb/s
tag6	NFC-A	ISO 14443-3A	Mifare Classic 1k	Yes	716B	106kb/s
tag7	NFC-A	ISO 14443-3A	Mifare Classic 1k	Yes	716B	106kb/s
tag8	NFC-A	ISO 14443-4	Mifare Classic 4k	No	~4kB	106kb/s
tag9	NFC-F (FeliCa)	JIS 6319-4	RC-S967	No	208B	212kb/s 424kb/s

Table 3: Inventory of PICC devices

On the PICCs that are marked writable, the content is harmonized to ensure the algorithm won't use the content as a feature to identify devices.

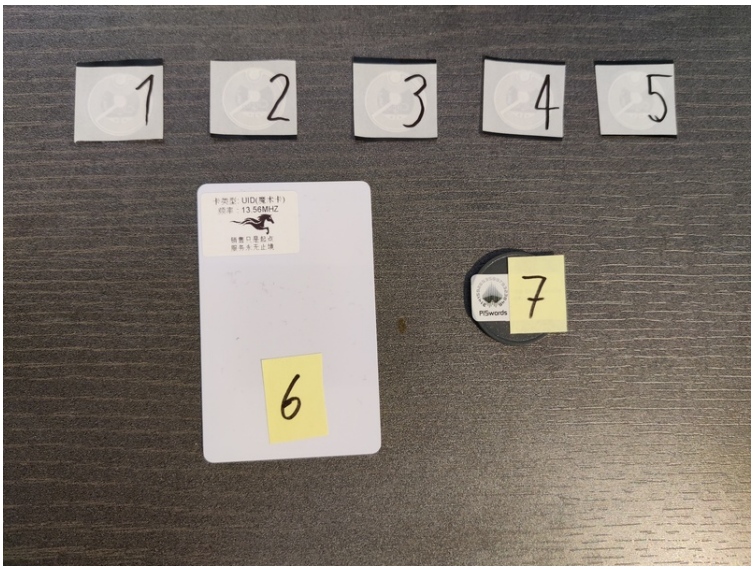

Figure 8: NFC tags 1-7

7.4 Acquisition script

In order to make the acquisition process simpler, we created a very simple acquisition script based on a script generated by GNU Radio Companion.

The goal is to make it as easy as possible to start recording with a selection of parameters, and to record for a set amount of time (a set amount of samples to be precise). To do this, we use the "Osmosdr source" block to connect to our Airspy HF+ device and the "Head" block to set a fixed amount of samples until the script stops.

The script is written for the Airspy HF+, but is very easy to modify for one's needs. The sampling rate, the center frequency and the capture length are all optional arguments. A path for the output file must also be specified.

7.5 Dataset description

This section describes the two main datasets which were built for this project using the setup described higher in this section. Both are available through git LFS in the same GitHub repository as the source code for this project.

7.5.1 First dataset

In our first attempt at building a dataset, we tried to make things simple. We recorded three communications between the reader and each of the nine tags. This was so we could compare the signals and their features between different tags but also between different captures of the same tag. The acquisition started after the communication was already established, so the transient part of the signal is absent.

As the list below shows, the captures each stopped after a fixed amount of samples were received. We used the script described in section 7.4 to simplify the process and minimize variability. The parameters passed to the capture script were `--samplerate 768000` and `--time 3`.

- 3 recordings of each of the 9 tags
- Sample rate: 768 kS/s
- Length of a recording: 3 seconds
- Size of a recording on disk: 18.432 MB
- Total size of the dataset: 497.664 MB
- (Content of tags 1 through 7: 36B of the 'A' character.)

Just for visual reference, we include a representation of an NFC transaction between reader and tag for each tag type (NTAG, MiFare, FeliCa chip). Figures 9, 10, and 11 show us the difference between these types in a regular request-response.

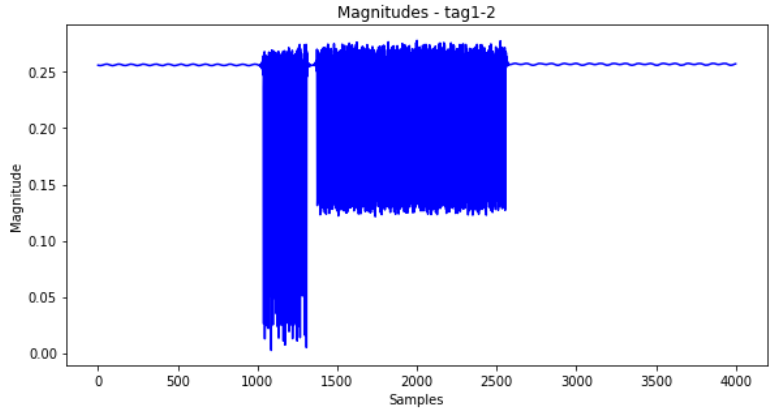

Figure 9: NFC transaction of an NTAG213 chip

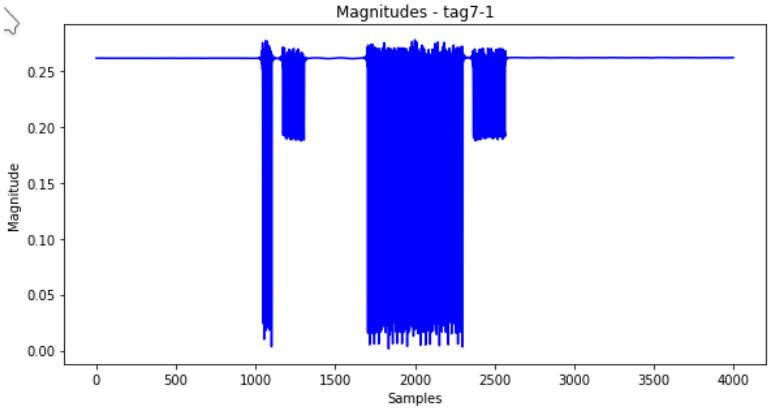

Figure 10: NFC transaction of a MiFare chip

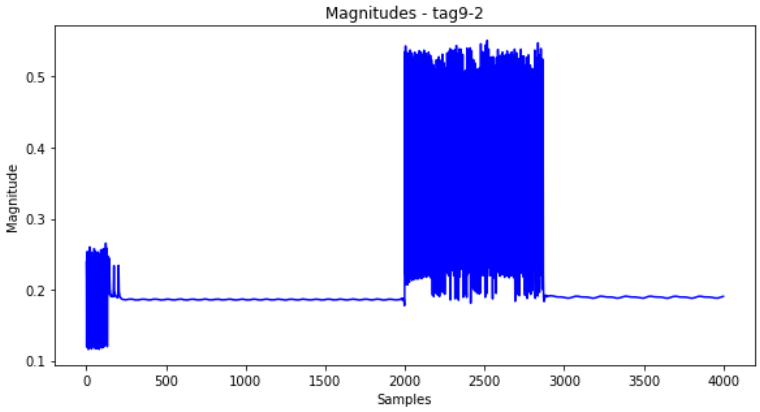

Figure 11: NFC transaction of a FeliCa chip

7.5.2 Second dataset

This dataset was built in the same manner as the previous one but the captures are much longer and more variable. We were also able to increase the sample rate thanks to a firmware update of the Airspy HF+. The goal here was to address the shortcomings of our first dataset in terms of volume and variability. To achieve that, we made the capture time longer and also captured the very start of the communications: the transient part.

The reader was held at a distance of approximately 1.8cm from the tags. We also removed tag9 from this dataset since it is so different from the NFC-A tags.

- 3 recordings of each of the first 8 tags
- Sample rate: 912 kS/s
- Length of a recording: 20 seconds
- Size of a recording on disk: 145.92 MB
- Total size of the dataset: 3502.08 MB
- (Content of tags 1 through 7: 36B of the 'A' character.)

As with the previous dataset, this one was built with the help of the capture script with the parameters `--samplerate 912000` and `--time 20`.

7.6 Validating the dataset

Because the dataset is such a critical part of the project, we wanted to make sure the capture contained the information we needed. The first idea to make sure it was the case was to decode the signal. The work on decoding is described in the next section, but we soon understood it wasn't enough.

Even if we could reconstruct the data, it didn't mean there was enough bandwidth or enough resolution to extract characteristics from the data. This is why we then tried to use measure tools like oscilloscopes to try and see if we could find tags' features with this highly sensitive hardware.

7.6.1 Decoding

The next step in the elaboration of the dataset is to be absolutely sure that the data is fully captured by our setup. The previous section seems to show this is the case, but to be sure we would need to decode the signal.

To do so, we need to know the modulation and coding used by the protocol. In the case of our NFC-A tags, the reader's transmissions are coded with a modified Miller code, while the tag's responses are coded with Manchester coding. Both modulate the data with On-Off Keying (OOK), which represents zeroes as no change of amplitude and ones as changes in amplitude over a given time period. [19]

Miller coding, as applied in NFC communications, works by mapping four symbols in the signal to a bit. A one is always coded as "high, high, low, high" or 1101. A zero can be coded as 0111 or 1111, depending on whether it came after a zero or a one respectively. [20]

Manchester coding on the other hand, uses transitions to express bits. High-to-low stands for a one and low-to-high represents a zero. This only takes into account transitions that happen at the middle of a period. Transitions at the start of a period don't matter. [20, 21]

Using Rona [22]'s GNU Radio Companion module `gr-nfc`[8], we were able to decode messages coming from the reader. An excerpt of the decoded requests issued to tag1 is shown in figure 12. They show typical NFC requests like 52 (wake up), 50 00 (HALT) or 93 and 95 which are anticollision requests. The figure also shows tag1's UID is present in the anticollision requests, using a screenshot of the tag's properties.


```
Reader -> 50 00 57 CD
Reader -> [26]
Reader -> 50 00 57 CD
Reader -> [52]
Reader -> 93 20
Reader -> 93 70 88 04 5A F5 23 B5 9A
Reader -> 95 20
Reader -> 95 70 2A 37 60 80 FD 50 13
Reader -> 50 00 57 CD
Reader -> [52]
```

Figure 12: Decoded frames from the reader, showing tag1's UID in red

Before we decided to use other means to verify the signal, we tried decoding with several tools and existing libraries. We tried demodulating the signal with python before feeding it to `sigrok`'s decoders[9], without success. We tried Universal Radio Hacker[10], a signal analysis application, but we didn't seem to find the right combination of parameters to get back our data. In the end, Roberto Rigamonti wrote a MatLab state machine that was able to decode single requests from the reader, showing that the gr-nfc module was right.

7.6.2 Measure tools

	Response delay	Rise time	Fall time	Width
tag1	85.82us	337-433ns (375.74ns)	356-538ns (417.15ns)	587-615ns (601.59ns)
tag2	85.85us	359ns-6.324us (488.3ns)	357-551ns (423.7ns)	513-569ns (541.1ns)
tag6	85.6us	356-454us (398ns)	350-444ns (393.44ns)	555-582ns (569.956ns)
tag7	90.8us	358-445us (402.88ns)	350-436ns (389.78ns)	591-610ns (601.346ns)
tag9	2.42ms	321-438us (348.78ns)	350-532ns (405.05ns)	2.376-4.787us (2.783us)

Table 4: Excerpt of the features observed on the oscilloscope

We were given access to precise and powerful measure tools such as the *N9020A MXA Signal Analyzer* from Keysight and the *WaveMaster 808Zi* from LeCroy. The goal was to determine whether there existed distinguishable features in the tags' signals that were very different from

[8] https://github.com/jcrona/gr-nfc
[9] https://sigrok.org/wiki/Protocol_decoders
[10] https://github.com/jopohl/urh

one tag to another. If we could find such features, we could then write software to extract them and it would make the problem easier.

Despite our efforts, we weren't able to find discriminative features using this method. Table 4 shows a few of our observations, just to highlight the fact that even for a single tag, the readings can differ quite a bit. In this table, the mean is shown between parentheses.

7.6.3 Features extraction

In the continuation of our search for meaningful features in the data, we used the `tsfresh` library for python. This library is built to analyse time series data and compute an impressive number of statistics and features. It is built for feature engineering, which is why it includes utilities able to determine the ability of a feature to discriminate between classes. To do this, they provide feature selectors that evaluate the link between a feature and a class using hypothesis tests and p-values. This is a very thorough way to mathematically define the relevance of a feature.

As can be imagined, running this tool on very large dataset is expensive in terms of computation and memory. To mitigate this, we first computed a subset of statistical features ("only" 1514 features) using the `EfficientFCParameters` class. We also tried giving the library smaller segments of our signals. In both cases, we successfully computed all the statistics and stored them in CSV files. Sadly the feature selection phase, in which `tsfresh` removes all the unimpactful features left us with nothing. It wasn't able to find relevant information in the computed features.

Despite our efforts, we weren't able to find relevant features, be it "manually" with the measure tools or through automatic means like `tsfresh`. With this said, as far as we are aware, neither one nor the other would be able to find features that exist in the relation between I and Q, like some of the features mentioned in the literature (e.g. I/Q imbalance). We will try to prove that the data we collected contains the necessary information to discriminate between tags, despite the lack of evidence in this section. It would be interesting to go further in this feature engineering endeavour in other works.

8 Data preparation

Once the dataset is captured and stored, the data still has to be formatted and prepared to be fed to our models. We decided to alter the recordings as little as possible, since it is very valuable to have raw data in projects like this one. This means any modification of the data is done at runtime. This section describes the formatting and preprocessing applied to the datasets.

8.1 Environment

We used python to work with our datasets and to write the models described in next section. Visualizations were done in `Jupyter notebooks` using `matplotlib`. Data processing was achieved in a python project using `numpy` to read and manipulate the data and `scipy`'s signal processing library.

8.2 Data partitioning

8.2.1 Raw partitioning

Our first approach with the datasets was to segment each recording in windows of size n. In order to be able to tune the window size and to see its impact on the performance of the model, we made it a parameter.

The result for each tag is an array of windows, each comprised of n complex samples. If the number of samples in a tag's recordings is of size L, then this array contains L/n windows. An example of a resulting window can be seen in figure 13.

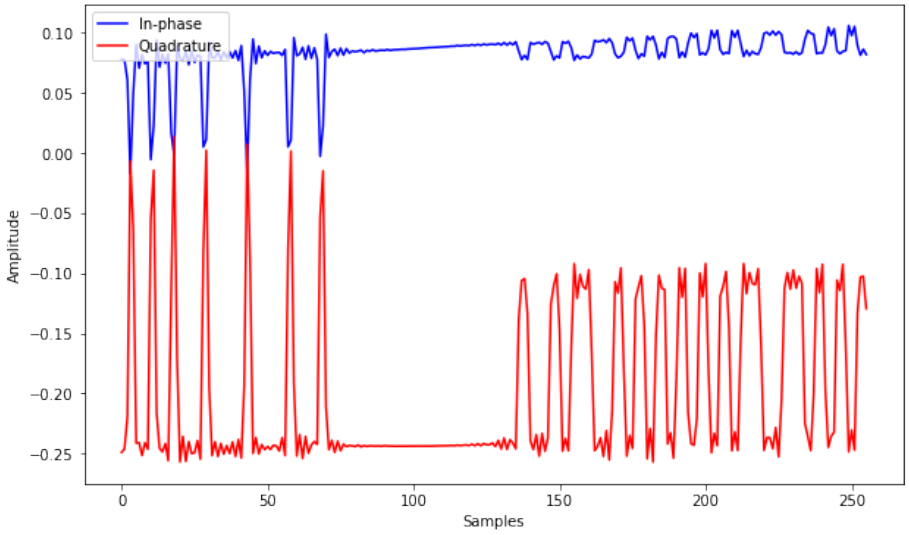

Figure 13: Visual example of a window of size $n = 256$

8.2.2 Detecting communications

It was soon clear that feeding the raw data without at least selecting segments that actually contained communications would not yield fantastic results. Indeed, most windows would contain only the carrier signal generated by the reader. These long segments of the signal contain

little to no meaningful information. This is why we used peak detection algorithms to detect actual communications and filter out the uninteresting parts of the signal.

When experimenting with the first dataset, we used the `signal.find_peaks` function from the `scipy` library. It allowed us to specify a minimum required height for a peak to be considered as well as the minimum required *prominence* of the peak. (The prominence defines the height of a peak relative to the lowest contour lines around it that don't include higher peaks [23].) This allowed us to detect, peaks from all of the transactions in the signal, with almost no false positives. We could then create our windows as decribed in the section above, using some peaks as the start of our windows, in a similar fashion as Youssef et al. [13] did in their work.

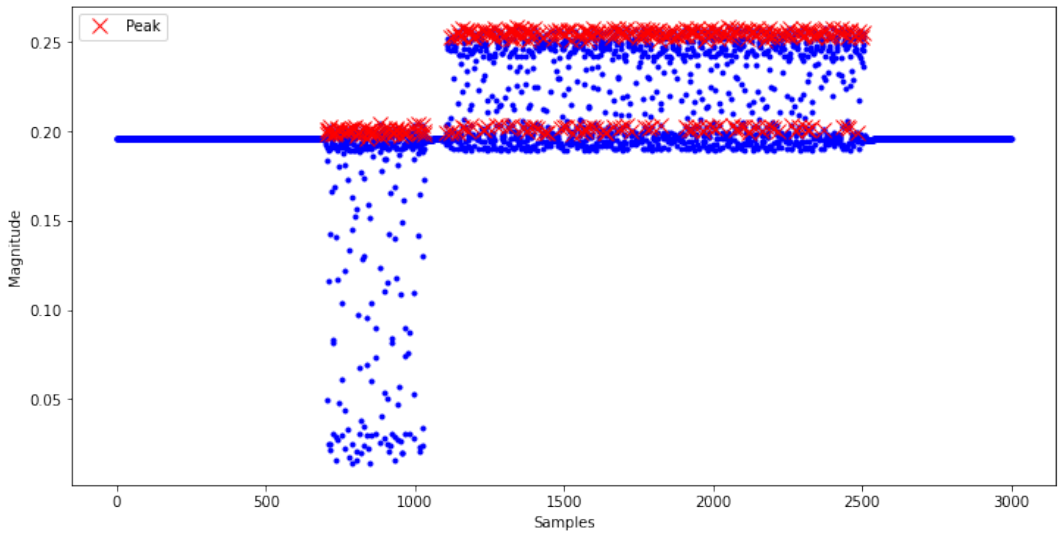

Figure 14: Visual representation of the peak detection on an NFC transaction

When our second dataset came into play, it became clear that calculating the prominence was computationally too expensive for data of this size. Using the more common *threshold* parameter, which is the vertical distance of a point to its neighbours, worked just as well for our needs and performed about 220 times faster (1min 21s vs 365ms, on our system). It should have been obvious from the description of these characteristics that prominence was a lot harder to calculate, but it didn't occur to us initially.

Using windows of length 256, the total number of windows in dataset 2 is 32'168 (4021 for each tag, as the number of window is balanced between classes). We considered augmenting our dataset using sliding windows which overlap, as Riyaz et al. [10] propose, but couldn't find the time in the scope of this project. This would considerably expand the dataset's size and potentially bring superior shift invariance to the model. This is an interesting idea to explore further.

8.3 Input formatting

Once our data is filtered and partitioned for input, the question of the format comes up. Indeed, a machine learning model will not accept complex numbers as input without adapting the loss function. This can be done without great difficulty, but most of the research we considered preferred representing the data as a two-dimensional array comprised of two float arrays: one for the real parts and the other for the imaginary parts. Figure 15 shows a representation of the input after the windows were created and formatted this way.

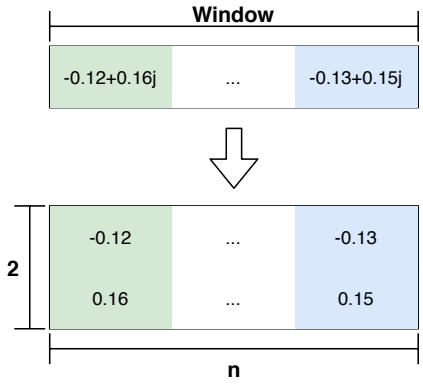

Figure 15: Format for the two-dimensional input

For our experiments with SVMs, such a representation wasn't appropriate since SVMs don't accept multidimensional inputs. This is why we propose two formatting options: "2D" for the neural networks and "1D" for the SVM. The latter appends the array containing the imaginary parts to the array containing the real parts, as shown in figure 16.

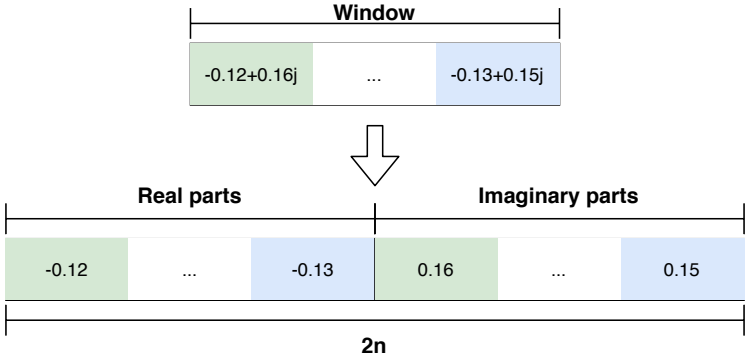

Figure 16: Format for the simple input

8.4 Normalization

Normalization is the action of changing the scale of numerical values in a dataset to a common range. It is essential in some machine learning applications, as it ensures one type of features won't have a greater effect on the different parameters of a model than another.

Of course, we don't have many different features with different scales in our dataset, so normalization is not absolutely essential here. However, there are amplitude differences between our recordings and making sure every sample stays in a set range will help the neural network

converge faster. Here, we chose the range $[-1, 1]$ for our values, as they already oscillate around 0 by nature.

8.5 Splitting into training, testing and validation datasets

Another very common operation in data preparation for machine learning purposes is randomly splitting the data into three separate sets.

- The *training set* is usually the biggest chunk of the dataset. It is fed to the model in batches for it to train itself to associate an input to an output.
- The *validation set* is the data we can use to monitor the performance of the model at each training iteration. We give this data to the training algorithm, but it doesn't use it to train the model. Instead, it uses it to validate the model's performance after each epoch. We can use the information given by the validation predictions to tweak the model's parameters. This is also the data we monitor to perform "early stopping".
- The *testing set* is used once the model is completely trained. As it has never been used before, neither by the model, nor by you before you tweaked the parameters, it can be used as an unbiased benchmark of the finished model.

In this project, we chose to separate our dataset as follows: 70% for the training set, 20% for the validation set, and 10% for the testing set.

9 Machine learning

9.1 Environment

In addition to the previous section's list of tools and libraries, we can mention `scikit-learn` whose SVM's implementation we use for our experiments. We also use its utility functions for data splitting and for metrics.

To define and train our neural networks, we use `keras` with `tensorflow` as the backend.

On our computing server, equipped with an NVidia GTX1070 graphics card, we use `tensorflow-gpu` to accelerate the calculations by using said graphics card.

9.2 Model architectures

For all of the models described hereafter, we have made the input dimensions and the number of outputs modular. The goal is to be able to build different experiments easily and compare performance.

First, we use the Support Vector Machine (SVM) model provided by `sklearn` as a "control experiment". If the SVM's performance is as good or better than our more advanced models, we can worry that something is wrong. Either the problem is too easy, which probably means there are biases in the dataset, or our other models are not appropriate for the task.

Our first attempt at a Convolutional Neural Network (CNN) is based on the description in the work of Riyaz et al. [10]. A representation of the layers is shown in figure 17. It achieved reasonable performance but was very prone to overfitting. Despite adding l2 regularization and increasing the dropout rate, this model still showed significant overfitting.

We later discovered that this was linked to the layer on which we performed the dropout operation. Indeed, it seems that in this situation, dropping out after the pooling layer, as opposed to doing it after the convolution layer, is a lot more effective. At this point, we had already started using the next CNN, which gave us slightly better results.

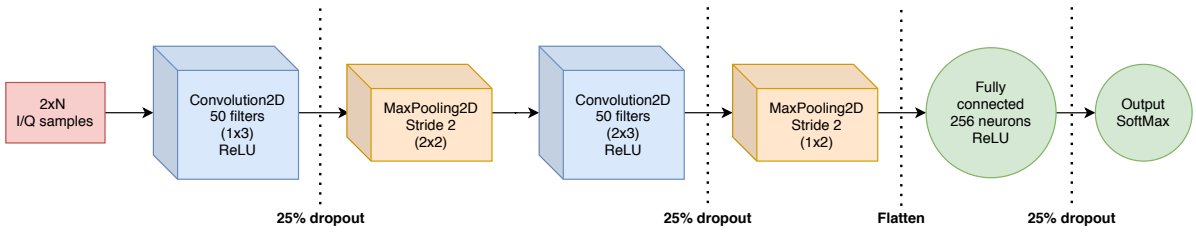

Figure 17: Representation of the first CNN used

Said CNN is based on Youssef et al. [13]'s description of their model. It is very similar to the previous one, but the convolution parameter seem more sensible and gave better results in our case. A representation of the model's layers is shown in figure 18.

We tried adding an additional convolution layer to improve our accuracy but the performance boost was insignificant and of course it made the model slower to converge. The same applied when adding an additional dense layer. Increasing the number of neurons in the dense layer had a similar effect and also seemed to increase overfitting.

So it seems like this simple model is hard to improve upon. In the end, we added the dropout effect to the dense layer in addition to the pooling layers, in order to lower the tendency to overfit even more.

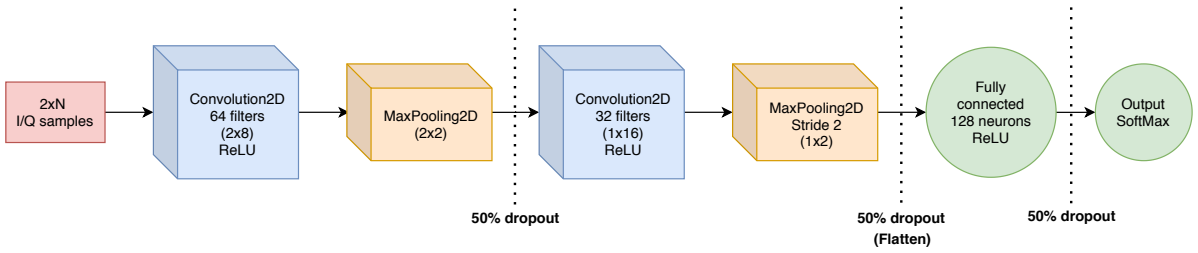

Figure 18: Representation of the second CNN used

9.3 Default parameters

Except if stated otherwise, the following experiments will use the model parameters listed in table 5.

Parameter	Value
Window size	512
Optimizer	Adam
Initial learning rate	0.001
Loss function	Categorical cross entropy
Batch size	500
Number of epochs	200

Table 5: Default model parameters for our experiences

9.4 Documenting experiments

9.4.1 Discriminating between chip types

The first experiment we report on is about classifying tags from different types. To do so, we use tags 1, 6 and 9 from the first dataset. The first is an NTAG213, the second a MiFare 1k and the last a FeliCa chip. As the first dataset is quite small, we use it in its entirety, rather than applying the peak detection algorithm on it. Our SVM attained an accuracy of 69%, and the confusion matrix is shown in figure 19.

At this task, our CNN performed slightly better with an accuracy of 74%. Its confusion matrix is shown in figure 20.

As we had anticipated, discriminating between our NFC-A tags (the NTAG and the MiFare) and the FeliCa tag is trivial, but finding the difference between NFC-A tags is a lot more difficult. Here the SVM probably does little better than random guessing for the first two classes.

This experience is what motivated us not to include the FeliCa tag in the second dataset. It shows that its presence doesn't add any value to the experiments.

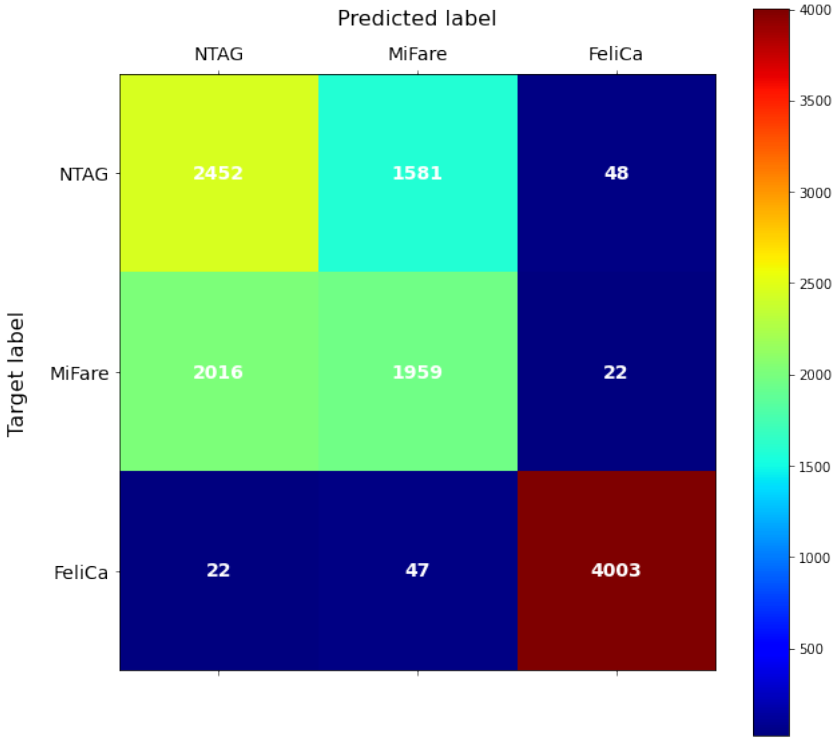

Figure 19: Confusion matrix for the first experiment on SVM

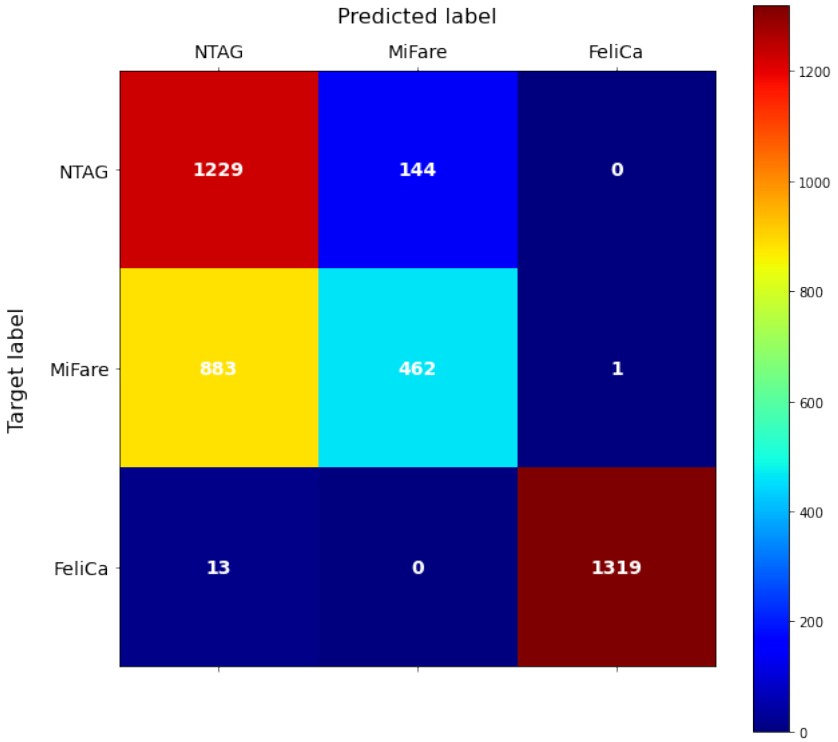

Figure 20: Confusion matrix for the first experiment on CNN

9.4.2 Discriminating between tags

Here, we use the second dataset with the peak finding algorithm to evaluate the performance of our model in the scenario that interests us the most: discriminating between individual tags. Is our model able to extract features to fingerprint the devices?

In figure 21, we compare the accuracy of our model for different number of tags with the accuracy of the SVM for the same numbers of tags. Of course, such a simple summary doesn't capture the full picture, but it gives a good idea of the performance of each model in different conditions. We ran the training several times and confirm that those scores are consistent.

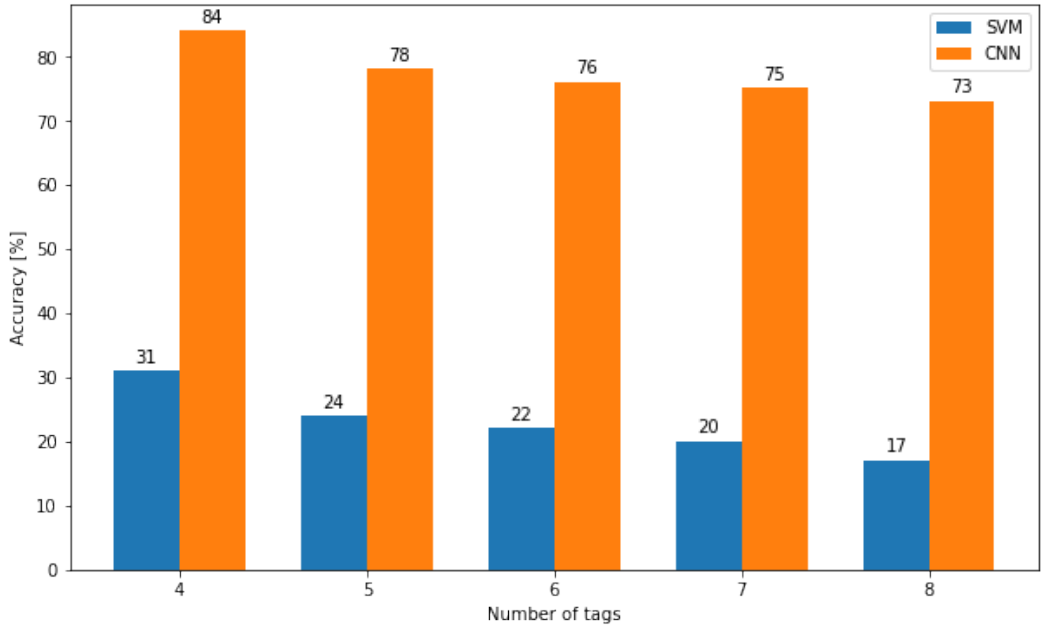

Figure 21: SVM and CNN accuracy per number of tag

We'll now observe the performance of our CNN when trying to discriminate between our eight tags. For this experiment, we increased the window size to 640, and found that it consistently gave slightly better results in this scenario. Continuing to increase the window size past this number does not help the performance.

In this experiment, we got an accuracy of 76%. The confusion matrix is detailed in figure 22 and we can see how close together the training accuracy and the validation accuracy are in figure 23. This indicates a welcome lack of overfitting with these parameters.

We also observed the model after 500 epochs to make sure it didn't continue to improve after the 200 standard epochs. This is not the case as the validation accuracy locks at around 75% accuracy, while the model starts to overfit slightly.

9.5 Thoughts on the results

76% top accuracy is not a very high number, but it is high enough to prove the network is able to extract features which belong to an individual tag. On the other hand, it is not so high as to show that the problem is trivial, or that we have made it trivial by mishandling some element of the pipeline.

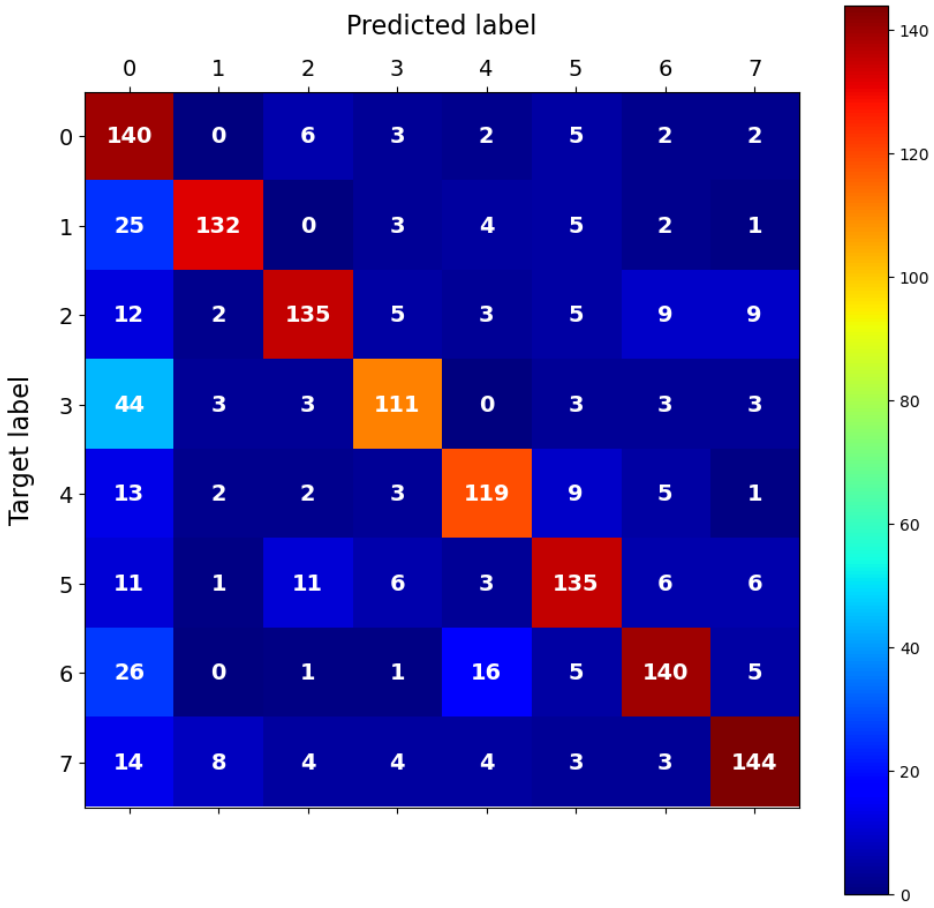

Figure 22: Confusion matrix for the CNN experiment

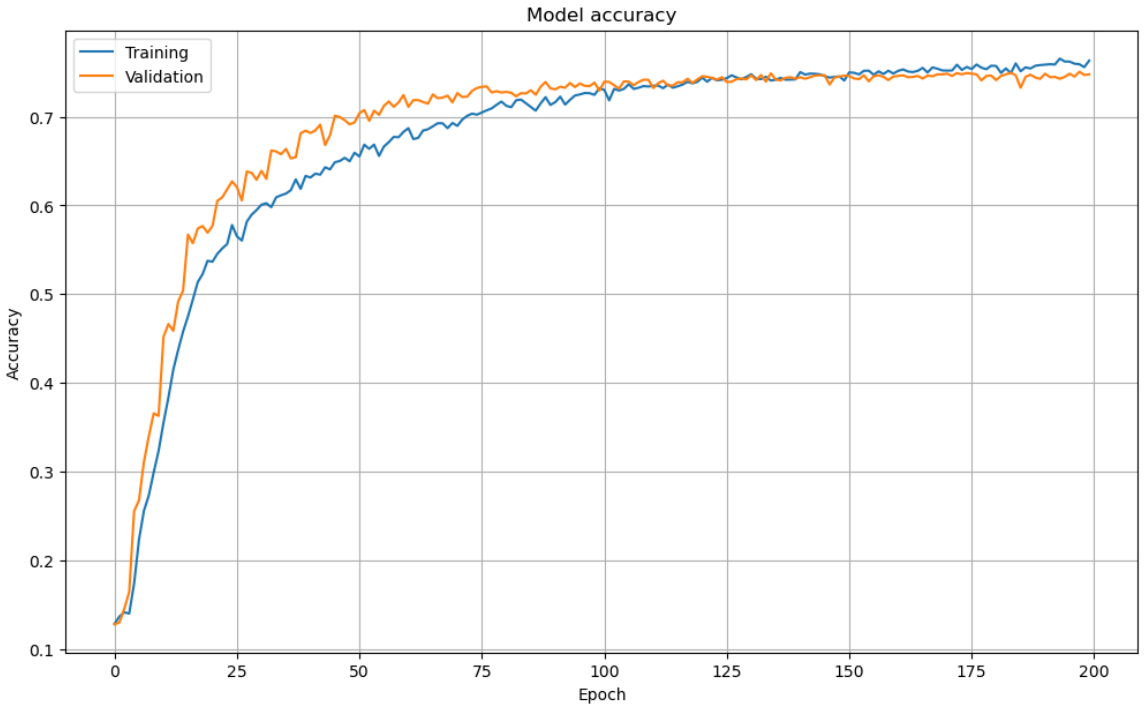

Figure 23: Accuracy history for the CNN experiment

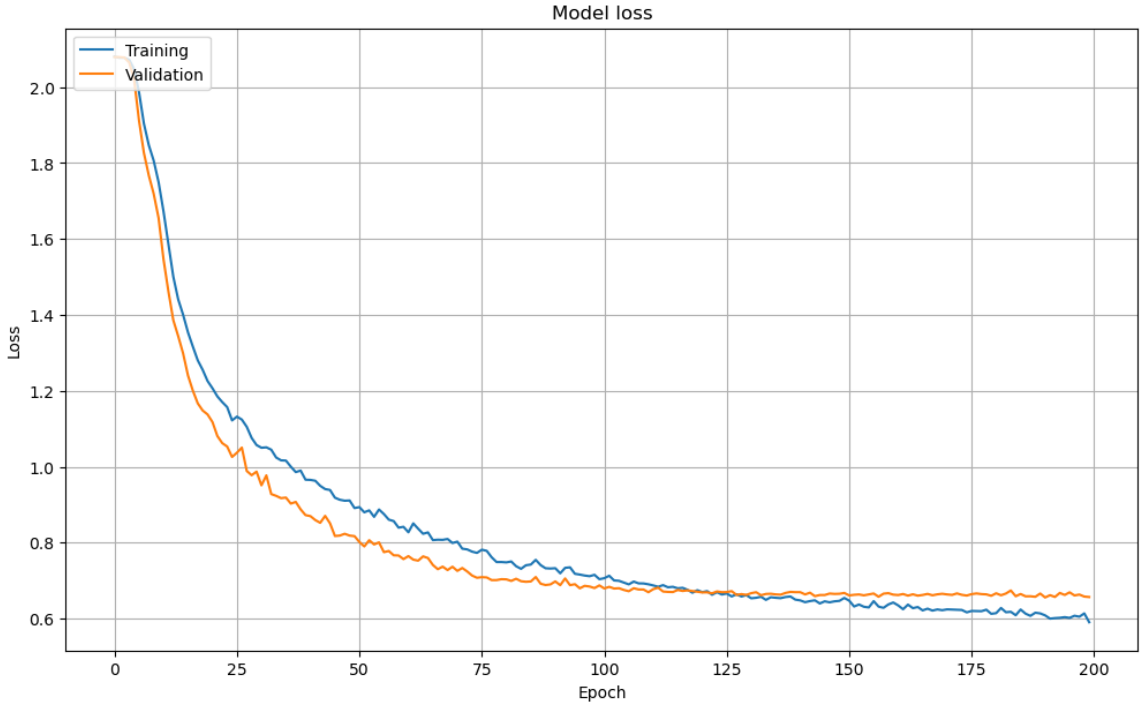

Figure 24: Loss history for the CNN experiment

The fact that the SVM performs barely 5% better than a random guess ($1/8 = 0.125$) in the most complicated scenario shows that the problem is hard. Any bias we could have introduced in the acquisition phase doesn't seem to make the problem easier. Despite this, our CNN is able to solve the problem, at least to some extent.

Furthermore, all the confusion matrices and accuracy scores we have shown here come from the trained model making prediction on the testing set. It has never seen those specific signal windows during training, which is another indicator that the features extracted by the convolution layers are meaningful.

Finally, as we could see from figures 23 and 24, our CNN with these parameters isn't really prone to overfitting. The validation accuracy doesn't stay down while the training accuracy increases, as was the case with the first CNN we used. One last positive indicator is the stability of the accuracy and the loss. There is no rough movement up and down, which means the adam optimizer is able to do its job without trouble.

All these points tend to show that we met our goal, at least in some proportion. However, the numbers tell us that this is not a scalable solution. Figure 21 clearly shows how quickly the accuracy drops as the number of devices increases. This is consistent with the findings of some of the papers we considered for this thesis. This is why we mentioned alternative ways or additional steps in the pipeline at several points in this report. It is important to go further and inspect additional techniques to refine the process in the future. This field is very new and very active at the time of writing, exciting evolutions will happen.

10 Conclusion

During this project, we have achieved tasks as varied as the elaboration of an acquisition setup for radio waves, attempts at decoding a modulated signal, learning about signal processing and data manipulation, and building deep neural networks.

Despite our best efforts to try as many things as possible, we have only scratched the surface of the theme of radio frequency machine learning. There are many elements we would have loved to explore: data preprocessing methods like wavelet transforms, dataset augmentation through sliding windows, complex weighted neural networks, unsupervised techniques, etc. These topics will have to be left to future work, as this project comes to an end.

We have learned a tremendous amount of useful knowledge in several complex domains and are satisfied with our thesis even though, as we said before, we would have enjoyed going further in analysing the results and improving the different aspects of the project.

Our results, while not overwhelming, are positive and give us reasons to believe the task is feasible. We are able to give elements of answers and to open paths for future developments, which will hopefully give way to more scalable solutions.

This report, the source code, and the datasets are all available on the git repository mentioned in this document's introduction (4).

Bibliography

1. PRITCHARD, Mike. *elttam - Intro to SDR and RF Signal Analysis* [online] [visited on 2020-04-12]. Available from: https://www.elttam.com/blog/intro-sdr-and-rf-analysis/.
2. Software-defined radio. In: *Wikipedia* [online]. 2020 [visited on 2020-06-16]. Available from: https://en.wikipedia.org/w/index.php?title=Software-defined_radio&oldid=959182555.
3. WILLIAMS, Al. *Your First GNU Radio Receiver With SDRPlay* [Hackaday] [online]. 2015-11-12 [visited on 2020-04-12]. Available from: https://hackaday.com/2015/11/12/your-first-gnu-radio-receiver-with-sdrplay/.
4. SPIESS, Andreas. *#286 How does Software Defined Radio (SDR) work under the Hood? SDR Tutorial* [online]. 2019 [visited on 2020-02-24]. Available from: https://www.youtube.com/watch?v=xQVm-YTKR9s&list=PLRlusPRPpIXTMhPZPn0IT0Wx_UTDW6iwi&index=12&t=273s.
5. KUISMA, Mikael. *I/Q Data for Dummies* [online] [visited on 2020-04-10]. Available from: http://whiteboard.ping.se/SDR/IQ.
6. OSSMANN, Michael. *Software Defined Radio with HackRF - Great Scott Gadgets* [online] [visited on 2020-02-24]. Available from: https://greatscottgadgets.com/sdr/.
7. Near-field communication. In: *Wikipedia* [online]. 2020 [visited on 2020-02-21]. Available from: https://en.wikipedia.org/w/index.php?title=Near-field_communication&oldid=940679317.
8. XU, Qiang; ZHENG, Rong; SAAD, Walid, et al. Device Fingerprinting in Wireless Networks: Challenges and Opportunities. *arXiv:1501.01367 [cs]* [online]. 2015 [visited on 2020-04-24]. Available from arXiv: 1501.01367.
9. DELGADO, Oscar; KECHTBAN, Louai; LUGAN, Sebastien, et al. Passive and active wireless device secure identification. *IEEE Access* [online]. 2020, pp. 1–1 [visited on 2020-05-08]. ISSN 2169-3536. Available from: https://ieeexplore.ieee.org/document/9082628/.
10. RIYAZ, Shamnaz; SANKHE, Kunal; IOANNIDIS, Stratis, et al. Deep Learning Convolutional Neural Networks for Radio Identification. *IEEE Communications Magazine* [online]. 2018, vol. 56, no. 9, pp. 146–152 [visited on 2020-04-24]. ISSN 0163-6804 1558-1896. Available from: https://ieeexplore.ieee.org/document/8466371.
11. STANKOWICZ, James; ROBINSON, Josh; CARMACK, Joseph M., et al. Complex Neural Networks for Radio Frequency Fingerprinting. In: *2019 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)* [online]. Rochester, NY, USA: IEEE, 2019, pp. 1–5 [visited on 2020-02-24]. ISBN 978-1-72814-352-1. Available from: https://ieeexplore.ieee.org/document/8923089.
12. OYEDARE, Taiwo; PARK, Jung-Min Jerry. Estimating the Required Training Dataset Size for Transmitter Classification Using Deep Learning. In: *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)* [online]. Newark, NJ, USA: IEEE, 2019, pp. 1–10 [visited on 2020-03-13]. ISBN 978-1-72812-376-9. Available from: https://ieeexplore.ieee.org/document/8935823.
13. YOUSSEF, Khalid; BOUCHARD, Louis; HAIGH, Karen, et al. Machine Learning Approach to RF Transmitter Identification. 2017. Available also from: https://arxiv.org/pdf/1711.01559.

14. MORIN, Cyrille; CARDOSO, Leonardo; HOYDIS, Jakob, et al. Transmitter Classification With Supervised Deep Learning. *arXiv:1905.07923 [cs, eess]* [online]. 2019 [visited on 2020-03-13]. Available from arXiv: 1905.07923.

15. SANKHE, Kunal; BELGIOVINE, Mauro; ZHOU, Fan, et al. No Radio Left Behind: Radio Fingerprinting Through Deep Learning of Physical-Layer Hardware Impairments. *IEEE Transactions on Cognitive Communications and Networking* [online]. 2019, pp. 1–1 [visited on 2020-02-24]. ISSN 2332-7731, ISSN 2372-2045. Available from: https://ieeexplore.ieee.org/document/8882379.

16. NGUYEN, Nam Tuan; ZHENG, Guanbo; HAN, Zhu, et al. Device fingerprinting to enhance wireless security using nonparametric Bayesian method. In: *2011 Proceedings IEEE INFOCOM* [online]. Shanghai, China: IEEE, 2011, pp. 1404–1412 [visited on 2020-06-12]. ISBN 978-1-4244-9919-9. Available from: http://ieeexplore.ieee.org/document/5934926/.

17. *Our Review of the Airspy HF+: Compared against ColibriNANO, Airspy Mini, RSP2* [rtl-sdr.com] [online]. 2017-08-12 [visited on 2020-06-19]. Available from: https://www.rtl-sdr.com/our-review-of-the-airspy-hf-compared-against-colibrinano-airspy-mini-rsp2-rtl-sdr/.

18. MARKS, Peter. *AirSpy HF+ review - a nice SDR receiver* [online] [visited on 2020-05-13]. Available from: http://blog.marxy.org/2018/01/airspy-hf-nice-receiver.html.

19. On–off keying. In: *Wikipedia* [online]. 2020 [visited on 2020-03-06]. Available from: https://en.wikipedia.org/w/index.php?title=On%E2%80%93off_keying&oldid=937798188.

20. *NFC Physical Layer - Modulation & RF Signal » Electronics Notes* [online] [visited on 2020-06-19]. Available from: https://www.electronics-notes.com/articles/connectivity/nfc-near-field-communication/physical-layer-rf-signal-modulation.php.

21. Manchester code. In: *Wikipedia* [online]. 2019 [visited on 2020-02-21]. Available from: https://en.wikipedia.org/w/index.php?title=Manchester_code&oldid=931345727.

22. RONA, Jean-Christophe. *Sniffing and decoding NFC with a DVB-T stick (RTL-SDR) and GNURadio* [online]. 2017-10-15 [visited on 2020-03-06]. Available from: http://blog.rona.fr/post/2017/10/15/Sniffing-and-decoding-NFC-with-a-DVB-T-stick-rtl-sdr-and-GNURadio?pub=0#pr.

23. Topographic prominence. In: *Wikipedia* [online]. 2020 [visited on 2020-07-30]. Available from: https://en.wikipedia.org/wiki/Topographic_prominence.