University of Padova

Department of Information Engineering

Master Thesis in Ict For Internet and Multimedia

Physical Layer Jamming detection: a Machine Learning Approach

Supervisor
Prof. Stefano Tomasin
University of Padova

Co-supervisor
Prof. Stefan Valentin

Master Candidate
Matteo Varotto

Student ID
2037034

Academic Year
2022-2023

"Milano non è Milan, Italia è Milan"
— Zlatan Ibrahimović

Abstract

This thesis aims to illustrate the laboratory experience with initial goal using machine learning techniques to analyze the physical characteristics (i.e: International Standard Organization layer 1) of a wireless cellular channel in order to detect the presence of an attacker.
Thus, the expected outcome of the project is to construct a binary classifier, which takes as input information on the wireless channel and outputs the state of the channel through a binary classification: that is, whether the channel is in a state recognized as normal or whether it has been corrupted by the presence of an attacker.
Lab experiments were carried out using Software Defined Radios, both user-side and attacker-side. Therefore, the methodologies used to conduct these experiments are explained, specifying the theoretical background and commenting from a technical point of view on the results obtained.

Questa tesi si propone di illustrare l'esperienza di laboratorio con l'obiettivo iniziale di utilizzare tecniche di apprendimento automatico per analizzare le caratteristiche fisiche (cioè il livello 1 dell'International Standard Organization) di un canale cellulare wireless al fine di rilevare la presenza di un attaccante.
Il risultato atteso del progetto è quindi la costruzione di un classificatore binario, che prenda in input le informazioni sul canale wireless e fornisca in output lo stato del canale attraverso una classificazione binaria: ovvero, se il canale è in uno stato riconosciuto come normale o se è stato corrotto dalla presenza di un attaccante.
Gli esperimenti di laboratorio sono stati condotti utilizzando radio definite dal software, sia lato utente che lato attaccante. Vengono quindi illustrate le metodologie utilizzate per condurre questi esperimenti, specificando il background teorico e commentando da un punto di vista tecnico i risultati ottenuti.

Contents

Listing of acronyms

SDR . . . . . Software Defined Radio

IoT . . . . . Internet of Things

AI . . . . . Artificial Intelligence

ML . . . . . Machine Learning

NN . . . . . Neural Network

CNN . . . . . Convolutional Neural Network

AE . . . . . Auto Encoder

SVM . . . . . Support Vector Machine

MSE . . . . . Mean Squared Error

CAE . . . . . Convolutional Auto Encoder

DoS . . . . . Denial of Service

OFDM . . . . . Orthogonal Frequency Division Multiplexing

SNR . . . . . Signal to Noise Ratio

GPU . . . . . Graphical Processing Unit

LTS . . . . . Long Term Support

API . . . . . Application Programming Interface

CDF . . . . . Cumulative Distribution Function

RAN . . . . . Radio Access Network

1 Introduction

Cellular networks have become an essential part of modern society, transforming the way we communicate and so access information. Given the mass use of smartphones and other mobile devices, these networks have revolutionized the way people interact and connect with the world around them. Suffice it to say that until 30 years ago the idea of communicating remotely with other people via a connection that was not wired, such as landline telephones or early Internet connections via Ethernet, was unthinkable for an ordinary citizen.

The arrival of these devices in the early 1990s thus marked a turning point in the way people interfaced with the world, allowing them to access more and more information. Over the years and with the advancement of information technology, these devices have become more and more intelligent and capable of storing more and more information. Suffice it to say that in this decade a person through their smartwatch, a device about 5 centimeters in diameter, is able to receive calls, text messages, and pay at the supermarket.

Given the enormous potential of these technologies to transmit or receive information, they are not only used for civilian purposes, but also for industrial ones.

The advent of so-called IoT has made it possible to radically change the way factories are designed and conceived, making the production of them more efficient.

The market therefore for these technologies has been growing steadily in recent years, in fact:

- Experts expect the global IoT in manufacturing market size to grow from USD 33.2 billion in 2020 to USD 53.8 billion in 2025 at a Compound Annual Growth Rate of 10.1%. [1].

- Experts highlight that discrete Manufacturing, Transportation & Logistics, and Utilities industries will spend $40B each on IoT platforms, systems, and services.[2]

The advent, therefore, of these technologies in areas considered critical to a business or a governament has triggered the emergence of new attacks aimed at compromising the integrity of the proper functioning of these technologies.
Some examples of such attacks are:

- **Jamming**: is a tool used to prevent wireless devices from receiving or transmitting radio information.
  Jammers block the use of devices by sending jamming radio waves on the same band used to transmit information. This, for example, causes interference that inhibits communication between mobile devices and repeater towers, paralyzing all network within its range. On most cellular devices what appears during such jamming is simply a no-network signal.
  In fact, the smartphone interprets the incapability to transmit information as the absence of a cellular network.

- **Covert channel**: is defined as any communication method used to communicate and/or transfer information in a covert and stealthy manner. The primary purpose in using a covert channel is to overcome the security policies of systems and organizations. There are multiple types of cyber threats that can affect the multilevel security (MLS) of ICT infrastructure and systems, and they are increasing daily at an impressive rate. Any shared resource as a bandwidth of a spectrum can potentially be used as a covert channel and this makes everything more difficult.

After briefly describing the possible attacks that can be carried out, it is easy to deduce that all companies that want to remain competitive in the market need to develop prevention systems for these types of attacks, so that a malicious attacker cannot jeopardize, for example, the continuity of a factory's production.
This aspect turns out to be very important when entering the economic-business world, since in the event of a stop in production, the damage done to the company itself can be considerable. Without going into technical-economic details, for example, in March 2022 a Toyota facility in Japan was cyber-attacked and the production was affected for more than a day and influenced about 10000 vehicles, which is equivalent to 5% of the production of a month of the group in Japan.[3] This then primarily explains the reason why the previously described project took place in the first place, which is to try to study more robust solutions than those currently in the literature that can go about detecting possible attacks and intruders present in a wireless cellular network.

2

Wireless channel

2.1 Jamming on a wireless channel

3
Machine Learning

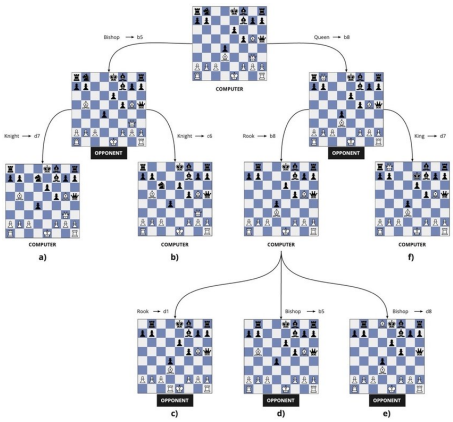
3.1 Introduction

The term Artificial Intelligence (henceforth AI) was invented by John McCarthy in 1956, at a two-month seminar (which he organized at Dartmouth College in Hanover, New Hampshire, USA) that had the merit of acquainting 10 U.S. scholars (on automata theory, neural networks and intelligence) with each other, and of giving the imprimatur to the term "Artificial Intelligence" as the official name of the new field of research.[4]
Since then, AI has established itself and evolved; today it is recognized as a branch autonomous, although connected to computer science, mathematics, cognitive science, neurobiology and philosophy.
Artificial Intelligence therefore represents a field of research and development that aims to create systems that are able to emulate and automate some human cognitive functions.
AI then is the ability of a machine to mimic some of the human cognitive functions, including machine learning, reasoning, planning, sensory perception, natural language understanding, and social interaction. AI can be divided into two main categories: **weak AI**, which focuses on specific and limited tasks, and **strong AI**, which aims to develop a machine with general intelligence comparable to human intelligence.
These goals nowadays are reached in two ways:

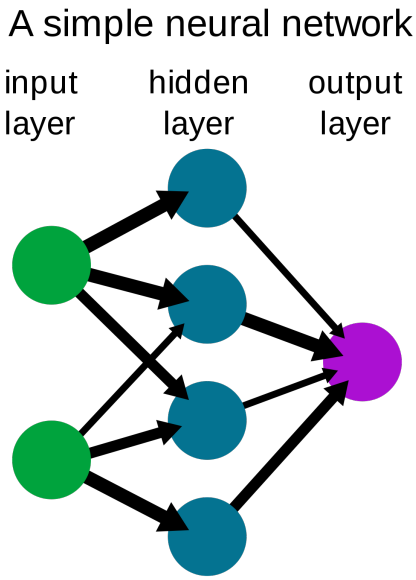
- As a first approach, AI can be created by telling to a model the rules to follow to solve

problems or to take decisions.
This is the case with the implementation of a machine attempting to play chess: here the chessboard is modeled and each possible move in the subsequent rounds is ranked according to its quality (via trees for example; see figure below) by predetermined heuristic algorithms, and the machine is required to choose the move that is rated most convenient, so as to maximize the probability of victory.
It is clear then that only simple-modelled problems can be solved by setting some rules to make the machine behave as a human; when it comes to deal with more complex structures (such as images, videos or texts) this approach shows its limits.

- The most common way nowadays to deal with more complex data structures is the so called **Machine Learning**, a branch of AI that allows machines to learn from data without being explicitly programmed to solve that specific problem, thus making a model trainable to solve multiple problems.
  It is easy to see that ML radically changed the approach to solve problems: whereas before we tended to study a problem to find the rules for solving it, now we tend to create a model to which we feed solved examples of that problem so that it learns to solve that problem on its own.
  The machine learning literature nowadays is widely developed and includes various approaches, ranging from a simple linear separator such as the perceptron to the more modern and now commonplace artificial neural networks, which allow extremely complex data structures to be handled and have extremely high performance.
  Without going into details (which will be explained later), it is possible to describe NNs as models that try to replicate the behavior of the human brain: a set of neurons (main elements of the network) are connected in order to take a datum as input and make a prediction or a decision attempting to minimize an error that is specified a priori during the training phase.
  A simple example will be shown below.



(a) Example of a decision tree for chess playing: in this case, the AI analyzes the current situation on the chessboard, expands some possible moves and basing on some given heuristics it computes the goodness of a choice.



(b) Example of a simple feed forward NN: this model takes its input throught the input layer, processes it in the inner layers and then outputs the result. In contrast of the example of the chessboard, the model has no pre-loaded heuristics to take a decision, but it has to be trained on several samples.

Figure 3.1: Two examples of AI.

As it was said before, the goal of the project was to apply a machine learning model in order to detect a jamming attacker; to understand the reasons why some models have been preferred to other ones, the most common ML models will be briefly explained.

3.2 Perceptron

The perceptron algorithm is a simple supervised learning problem that tries to solve a binary classification problem of linearly separable data.
The input of the model is a set $\mathbf{X} = (x_1, ..., x_n)$ of data and $\mathbf{Y} = (y_1, ..., y_n)$ of the corresponding labels. The goal of the algorithm is to find a vector $\mathbf{W} = (w_1, ..., w_n)$ of weights representing a separating hyperplane such that the classification error is avoided. Whenever a classification error is found, the vector $\mathbf{W}$ is updated. [5]
Recall that data is linearly separable if, given the training set composed by $\mathbf{X}$and $\mathbf{Y}$ and the

halfspace defined by $(\mathbf{w}, \mathbf{b})$ we have:

$$\forall i : (< w, x_i > + b) y_i \geq 0$$

i.e.: it perfectly separates all the data in the training set.
Here below a pseudocode of the iterative algorithm:

Algorithm 3.1 Perceptron algorithm

Require: $\mathbf{X} = (x_1, ..., x_n), \mathbf{Y} = (y_1, ..., y_n)$
  $w_1 \leftarrow (0, ..., 0)$
  for $t \leftarrow 1$ to ...
    if $\exists i$ s.t. $< w_i, x_i > y_i \leq 0$
      $w_{t+1} \leftarrow w_t y_i$
    end if
  else
    return w_t
  end for

From the pseudocode above it is easy to see that this algorithm could be easy implemented in any high level programming language nowadays and it is also demonstrated that if data is linearly separable the algorithm will stop in a finite number of steps.[6]
This model, however, shows its limitations:

- First of all, the convergence is not guaranteed when it is not dealing with a linearly separable training set, which is possible in more complex data structures.
- This algorithm can output different solutions depending on the starting values of the vector $\mathbf{W}$.
- The output of this classifier can only be a binary classifier, and so cannot be used in multiple class classification problem.

3.3 Support Vector Machine

After recalling the definition of linearly separable data (see section above) before talking about SVM, it is useful to definine the concept of margin, which will be useful later.

Given a separating hyperplane defined by L:

$$L = \{\mathbf{v} :< \mathbf{v}, \mathbf{w} > +\mathbf{b} = 0\}$$

and a sample x, the distance between x and L is:

$$min\{||\mathbf{x} - \mathbf{v}|| : \mathbf{v} \in L\}$$

The margin then, is defined as the minimum distance between a sample and L. [7]
The closest samples are called **support vectors**.
Here below an example of how SVM works:

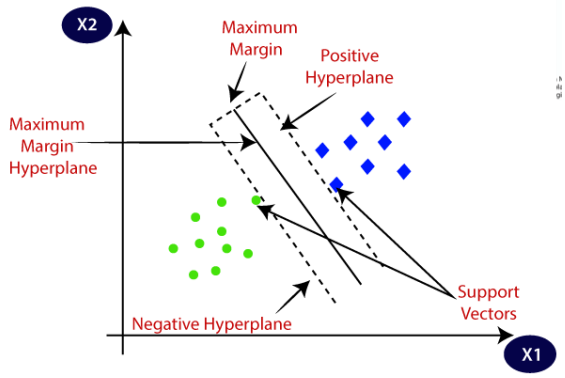


Figure 3.2: Representation of how SVM works in 2D dimensional space.

SVMs are also divided in two types:

- **Hard SVM**: a linear model that work with linearly separable data.
- **Soft SVM**: a linear model that work with non-linearly separable data.

3.3.1 Hard SVM

Hard SVMs seek for the separating hyperplane with the highest margin, under the assumption that the data is linearly separable. The mathematic formulation of Hard SVMs is expressed as follows:

$$\underset{(\mathbf{w},\mathbf{b}):||\mathbf{w}||=1}{argmax} \; \underset{i}{min} | < \mathbf{w}, x_i > +b \; |$$

3.3.2 Soft SVM

As mentioned earlier, hard SVM has the main problem that it assumes linearly separable data, which is impossible in most problems found in case studies.
Soft SVM therefore relaxes the previously set constraints taking into account their violation at the same time.
This implies in relaxing the constraints as follows:

- First of all, a set of slack variables are introduced: $\xi = (\xi_1, ..., \xi_n) : \xi_i \geq 0 \ \ \forall i$.
- For each $i = 1$ to n the constraint becomes : $(< \mathbf{w}, x_i > + b) y_i \geq 1 - \xi_i \ \ \forall i$.
- The model then tries at the same time to minimize the norm of w (to maximise the margin) and the average of ξ_i (to minimize the violations of the constraints).
- The objective function of the optimization problem then becomes:

$$\min_{(\mathbf{w},\mathbf{b},\xi)} (\lambda ||\mathbf{w}||^2 + \frac{1}{n} \sum \xi_i)$$

  Subject to the constraint defined before.
  It is clear then that a large λ makes the algorithm focus on the margin, while a small value of the variable makes the model to minimize the constraint violations.

This reformulation of the SVM model thus represents a way to solve a binary classification problem while having more complex data structures, however, which are not always linearly separable.[8]

3.4 Neural Networks

As mentioned earlier, neural networks are models whose structure is inspired by the functioning of an animal brain.
As a first general overview, a neural network can be regarded as a non linear mathematical function which transforms a set of input variables into a set of output variables. [9]
The network can me modeled as an acyclic graph $G = (V, E)$, divided into layers :

- The vertexes of the graph are the **neurons**, which take in input the sum of the outputs of the connected neurons from previous layer weighted by the edge weights and applies to this result a simple scalar function called activation function.

- The edges of the graph connect a neuron to other neurons of the next layer; to each edge is associated a weight. This implies also that the edges are directed and the flow is performed only in one direction.

The computation of the output is done by processing the input at each layer and forwarding it to the next layer, until the output layer is reached.
The activation functions of the neurons are defined a priori and cannot be changed during the training phase, making so the weights the trainable parameters.
Given a training set composed by $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$, where X is the training dataset and Y is the set of labels associated to the data of X and given a loss function l, the goal of the NN is to find the optimal values of the weights in order to minimize the loss L computed on the training set, that is:

$$L = \frac{1}{n} \sum l(x_i, y_i)$$

To do that, the most common way nowadays is the so-called **backward propagation** algorithm: at each epoch, the loss of the output is computed, and after that, it is propagated backwards to the input. Using the gradients calculated during error back-propagation, the network weights are updated. The update rule for the weights is defined as follows:

$$w_{ij}{}^{(t)[s+1]} = w_{ij}{}^{(t)[s]} - \eta \frac{\partial L}{\partial w_{ij}{}^{(t)[s]}}$$

Where $w_{ij}{}^{(t)[s+1]}$ stands for the weight at layer t computed at iteration $s + 1$ at the arc (i, j).
The gradient so is computed for each weight of the network, but after having computed the output at the last layer.
This procedure is repeated for many iterations, after a stopping criteria is reached, that could be:

- The maximum number of iterations.
- The reaching of a value in the training loss.
- The increasing of the validation loss for n epochs. (Recall that the validation set is a set with the same distribution of the training set but on which the NN is not trained; it is often used to set hyperparameters the ML models).

The simplest example of NN that can be shown is the so called *feedforward*: in this case each neuron is connected to every neuron of the next layer and the output is propagated from

the input layer throught the hidden layers and then to the output layer; an example was shown above in figure 3.1.b.
Here below will be shown a graphical representation of how a neuron works:

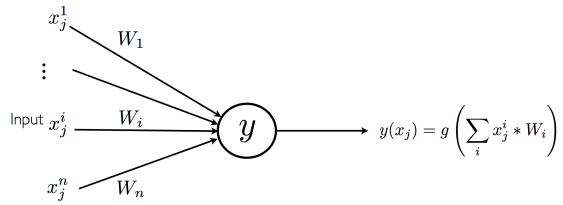


Figure 3.3: Representation of how a neuron works.

3.4.1 Convolutional Neural Networks

A convolutional neural network is one of the most significant networks in the deep learning field. Since CNN made impressive achievements in many areas, including but not limited to computer vision and natural language processing, it attracted much attention from both industry and academia in the past few years.[10]
The CNN is a kind of feedforward neural network that is able to extract features from data with convolution structures. Different from the traditional feature extraction methods, CNN does not need to extract features manually.
The main problem that occurs in feedforward NNs is the fact that the model does not take into account that a neuron can be more related to others and less to other ones; in fact, in this case every neuron is connected to each neuron of the next layer, without taking into account any kind of correlation.
CNNs then use some structures to achieve feature extraction:

- **Local connections**: each neuron is no longer connected to all neurons of the previous layer, but only to a small number of neurons, which is effective in reducing parameters and speed up convergence.
- **Weight sharing**: a group of connections can share the same weights, which reduces parameters further.
- **Downsampling dimension reduction**: a pooling layer harnesses the principle of image local correlation to downsample an image, which can reduce the amount of data while retaining useful information. It can also reduce the number of parameters by removing trivial features.

These three appealing characteristics make CNN one of the most representative algorithms in the deep learning (see definition below) field.
Pooling layers were at first used to reduce computational complexity, but turned out to be crucial to improve performance in many applications since they increase the receptive field of the inner layers. One of the most common pooling layer is the max-pooling, that it takes in input a window of samples and outputs the highest value of it.

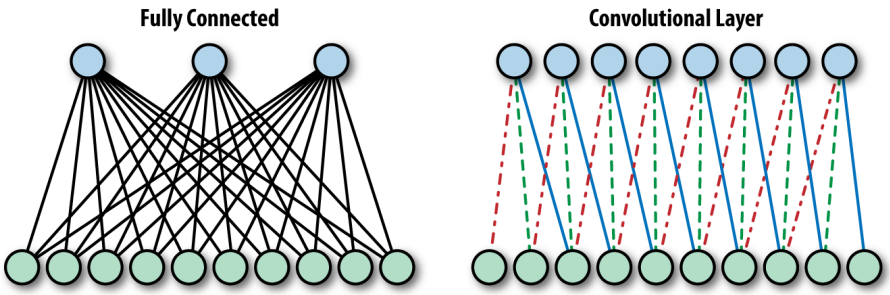


Figure 3.4: Graphical representation of the difference in terms of connections between feedforward NNs and CNNs.

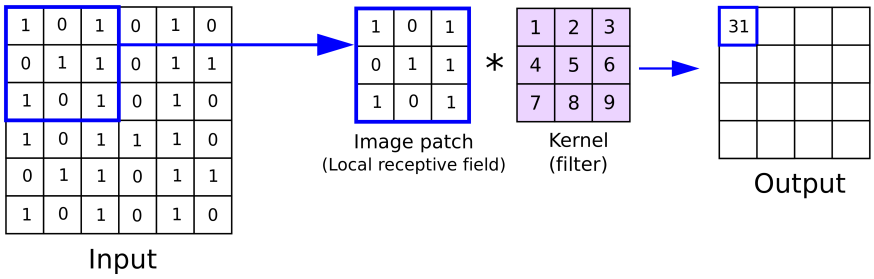


Figure 3.5: Graphical representation of how a convolutional filter can be viewed to process a 2D matrix.

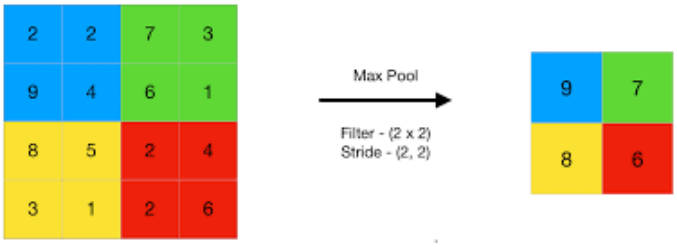


Figure 3.6: Graphical representation of a max pooling layer applied to a 2D matrix.

From figure 3.5 it is easy to see how convolutional filters can be represented as windows that slide along the input data.
To design a better CNN, it will be very important to set properly the **size** of the windows and their **stride**: in a Convolutional Neural Network, the "stride" refers to the step size at which the convolutional filter moves across the input data during the convolution operation. A larger stride value means the filter skips more pixels as it moves, resulting in a smaller output feature map with reduced spatial dimensions. Conversely, a smaller stride value leads to a larger output feature map with more detailed spatial information. Stride is a hyperparameter that can be adjusted when designing a CNN architecture to control the trade-off between spatial resolution and computational complexity.
The term **deep learning** was previously mentioned: it is a branch of machine learning that defines all models of NNs that have many layers.

3.5 Training, Validation and Test Set

As it was said before, machine learning models need a training phase to work; the training rules for each model are described above, but in this section how to create properly the **training**, **validation** and **test** sets in order to achieve better performances. [11]

- The training dataset contains the sample of data used to fit the model (wheights and biases in the NN case). In this case, so, the model **sees** and **learns** only from this data.

- At each **epoch** the model tries to minimize a loss function on all the samples of the training set.

- The validation dataset are the samples of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
  It is also useful during the training of the model to avoid the overfitting problem, that is the case in which the model makes good predictions on the training set but not on the other sets, due to a loss of generalization. By setting a monitor on the loss on the validation set (recall that the model is not trained on validation data, and so a constant increasing of the loss on the validation set may indicate a loss of generalization) it is possible to avoid this case.
  The validation dataset has a size of 30% of the training set and it should follow its same distribution, and so it should present the same kind (and relative percentage) of samples as they were present in the training.
  Supposing the task of an hypothetical NN is to distinguish apples and oranges and the

training set is composed of 500 oranges and 500 apples, the ideal validation set is composed of 150 oranges and 150 apples.

- At the end, the test set are the samples of data used to provide an unbiased evaluation of a final model fit on the training dataset.
  The test set provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets).
  Usually the test set has a size similar to the validation set.

4

Jamming detection with machine learning

As mentioned in Chapter 2, the informations that can be obtained from a wireless channel at level 1 of the ISO/OSI stack are two:

- The power of the signal among a bandwidth, the so called **spectrogram**.

- The sampling of the signal in two coordinates (in phase and quadrature) along a time interval. The result of this operation leads to the realization of the so-called **i-q diagram**.

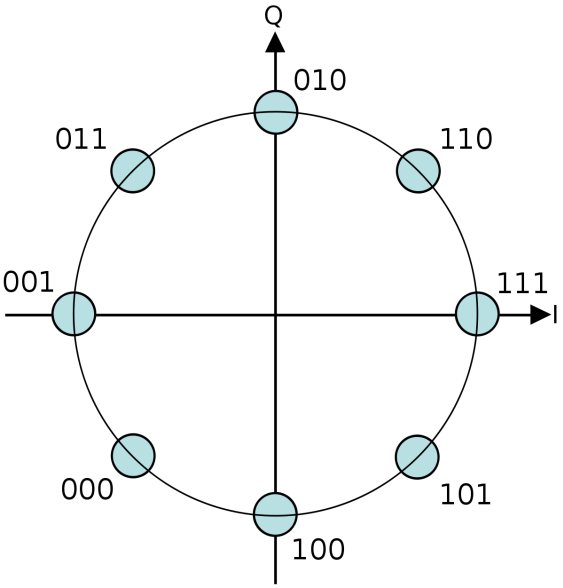


Figure 4.1: Example of an i-q diagram with 8psk modulation.

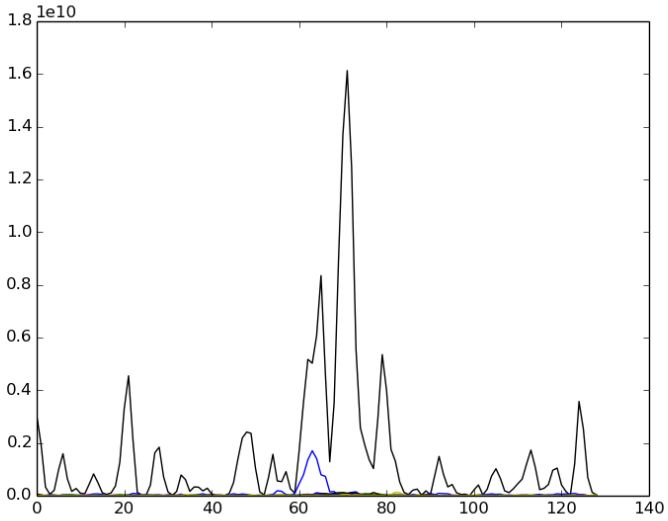


Figure 4.2: Example of a plot of a the power spectral density of a general signal among a bandwidth of 140 Hz.

The goal of the project, then, is to use this informations to create a model that can distinguish a channel in states deemed *trusted* from a channel corrupted by the action of a jammer.
This allows us to say that the goal of the project will be to solve a binary classification problem:

the output of the ML model will be the predicted class of the wireless channel, that will be: *trusted* or *jammed / anomal*.

4.1 Jamming detection on i-q diagrams

An i-q diagram is a collection of i-q samples over a period of time; the more the time resolution is high, the less there will be i-q samples in our plot.
To make it more clear, it is easy to think that for a given sampling rate collecting the double of the samples will take the double of the time.
After having determined the time resolution, it is necessary to decide the way to represent an i-q diagram in a computer; there are two ways:

- The first one is to represent our **n** samples as an array of tuples of length two (recall that each sample has two coordinates: in phase and quadrature), then use this data structure to classify the input data.
  This task can possibly be done by all of the ML models explained in Chapter 3.
- The second one is to plot the samples in a 2D plane and use the plot to classify the channel. This kind of problem can be easily linked to an image classification problem, which is well implemented in the state of the art by CNNs.

After having clear in mind of how the data can be represented, the next choice to be done is the ML model that will be able to solve the binary classification problem explained above.

- At first, the choice to use the perceptron algorithm was discarded, due to the high dimensionality of the input data (recall that it would be represented as an array of **n** samples and each sample will have two coordinates) and the not guaranteed convergence during the training phase.
- The SVM model was also discarded because it is reported that SMV with high dimensional data tends to use a lot of memory and loose performance in terms of accuracy. (remember to cite the paper, TBD)

The choice therefore fell between the standard feedforward NNs and the CNNs.
Due to the advantages of the CNNs explained above, it was chosen to represent data as actual i-q plots (images) and feed a CNN with that kind of data to solve the binary classification problem.
Using images to represent i-q plots also allows to train the same model on different time resolution cases, due to the fact that the images will have the same spatial resolution.

This will allow later to use the same model and compare 3 different time resolutions, which would not be possible if there were used other kind of data representations.
Modern literature shows two ways to solve this problem of image classification for jamming detection:

- The first one is the standard deep learning image classification model: it consists into taking as an input an image, apply several convolutional and pooling layers and end the NN with a final feedworward part that will have as the last layer **n** neurons, where n is the number of classes of the dataset.
  The output of the NN will be an array of probabilities that the input belongs to a class, and the predicted class will be the one with the highest score.

- The second one is to perform an anomaly detection problem using a deep learning model called **autoencoder**: an autoencoder is a deep learning model that takes an input data and by performing several operations in the inner layers tries to reconstruct the original input.
  The model is composed by two parts: the **encoder**, which takes the original input and transform it to a lower dimensional space data until it reaches the **latent space**.
  The **decoder** them takes as input the latent space and tries to reconstruct the original input.
  This kind of model implies that the classification is performed not on the labels of the input data, but on the reconstruction error (e.g.: MSE): is during the training phase the model is fed only with *trusted* images, it will learn how to reconstruct in a proper way only that kind of data, which implies that the so called *anomal* data will be reconstructed with an higher error. This implies also that the training phase is indipendent on the anomaly data.
  For the reasons explained above, the model choosen to perform the classification task on the cellular channel was the autoencoder.

By setting a threshold on the reconstruction error, the classification task between *trusted* and *anomal* data can be easily implemented, here below a pseudocode example:

Algorithm 4.1 Autoencoder classification task

Require: $X = INPUT\_DATA, Model = AUTOENCODER(), Threshold$
  $recon\_error\_x \leftarrow Model.compute\_reconstruction\_error(X)$
  if $recon\_error\_x > Threshold$
    return 0
  end if
  return 1

From this pseudocode it easy to see that with this approach is not possible to perform a multiple classification problem (even if the *trusted* cases belong to multiple classes), but it is possible to perform only an anomaly detection problem, which perfectly fits into the task of the project: in fact, it is better that the model knows only the normal behaviours of the wireless channel, in order to have a better response on attacks of different kind.
Here below an example of a simple CNN for image classification task and of an autoencoder:

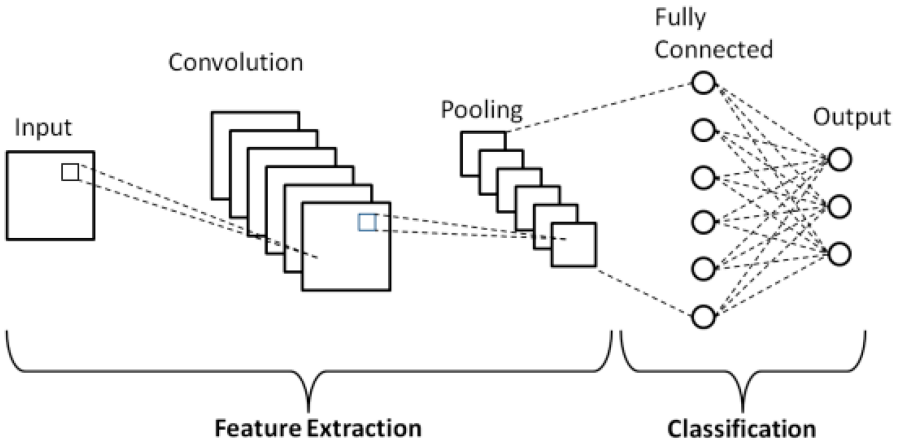


Figure 4.3: Example of a CNN for image classification: here then it is clear that the first layers are dedicated to the feature extraction task (pooling and convolutional layers) and then the NN in the last parts becomes feedforward and ends with n neurons, three in this case

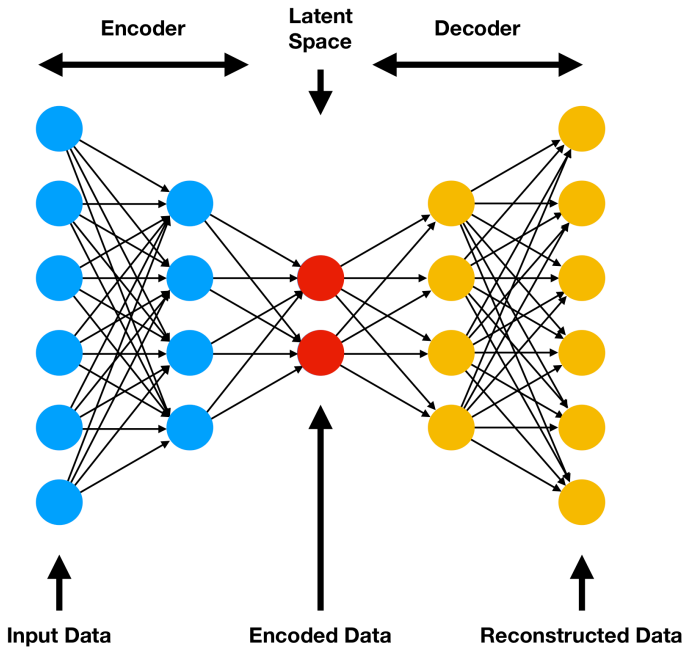


Figure 4.4: Example of an autoencoder.

Here above then is clear how the model reduces the input dimensionality to a latent space (output layer of the encoder) and then tries to reconstruct the original input.
For the reasons explained above, it was chosen to use a **Convolutional Auto Encoder** to perform anomaly detection on the images of the i-q plots, structured as follows:

- The first layers will be convolutional, in order to perform feature extraction on the image and reducing its dimension.
- Then, there will be some fully connected layers until it will be reached the latent space to add parameters to the network and so perform better generalization, due to the fact that convolutional layers contain less trainable parameters than fully connected ones.
- The structure of the decoder (from the latent space to the output layer) will be symmetrical to the one of the encoder, with some upsampling operations in order to have in the output layer the same dimension of the input layer.

Due to the fact that the network is trying to reconstruct an image, the chosen loss function will be the MSE, computed as follows:

$$SE(x,y) = \sum_{(i,j)\in M} [x(i,j) - y(i,j)]^2$$

$$MSE = \mathbb{E}[SE(x, y)]$$

Where:

- **M** is the input image, which is seen as a matrix from the model perspective.
- x and y are respectively the input image and the reconstructed image.
- i and j are the pixel raw and column indexes.
- $x(i, j)$ and $y(i, j)$ are respectively the value of the pixel in the input image and in the output image.

The details and the performance of this CAE adopted will be analyzed in the next chapter.

4.2 Jamming detection on waterfall plots

As it was said before, the other physical layer aspect that describes the wireless channel is the spectral power.
Using this aspect as input data, to perform jamming detection there will be created and analyzed the so-called **Waterfall Plots**.
Waterfall plots as a first approach can be seen as a sequence of PSD arrays stacked in order to create a matrix, built in this way:

- The X axis is the frequencies axis, centered on the carrier frequency, such as PSD arrays.
- The Y axis is the time axis.
- The value contained in the matrix represents the spectral density in a binned frequency range in a time slot.

Due to the description given above, it is clear that waterfall plots represent the evolution of the power spectral density over the time of the wireless channel. Using this kind of data structure, there will be explained the aspects of the channel that will be investigated in order to see if there is the presence of a jammer or not.
Waterfall plots can be represented in two ways:

- The first one is a raw matrix containing the power values for each cell of the matrix. This data structure is fast to compute, but is not so well human-readable.

- The second one is to convert the values of the matrix into a colorscale, which implies to convert the matrix into an image, usually RGB. This kind of data structure implies a better human-readability, but it involves the disadvantage of having an higher computation time, due to the fact that each information contained in the matrix has to be converted into a pixel of the output image.

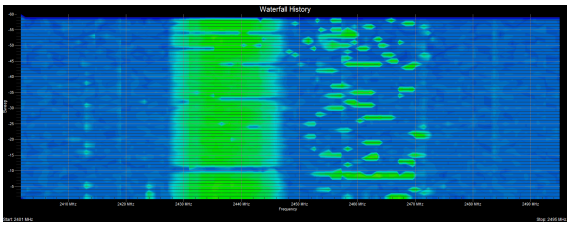


Figure 4.5: Example of a waterfall plot with the colorspace representation.

During the data creation phase, it was noticed that creating waterfall plots with the colorspace was more time consuming compared to creating raw matrixes with the raw power values inside each cell of the matrix.
In fact, creating a dataset with four thousands images took about 2 days, while creating a dataset with the same number of matrixes took about 15 minutes.
For this reason and due to the fact that both representations are 2D structures (and so the same ML structures can be used to analyze them), it was chosen to use a dataset composed by raw matrixes.
Due to the nature of the input data explained above, it was chosen to use also in this case a CAE, which is structured in a similar way as explained above in the i-q plots case.
While the model to analyze data is the same, the way how results will be seen is different. In the first case, the goal of the model was to distinguish between a trusted i-q plot and a jammed one; here instead the goal is to find out portions of the spectrum that result anomal according to the normal behaviour.
For example, in this case is not so meaningful to identify a globally jammed waterfall plot, because a smart jammer could act on a tiny portion of the spectrum, which in this case could lead a reconstruction error that could be below a given threshold. It will be more useful then to highlight the portion of the waterfall plot where the reconstruction error is high, highlighting then the corrupted parts of the spectrum.
It is also useful to explain a reasoning that was done before the data analysis on these plots: due to the fact that looking at the power in the mainband is not so meaningful to detect the

presence of a jammer, it is expected that the higher reconstruction errors will be present in the sidebands of the channel, where usually our device tends to leak less power.
To simplify this concept, it is useful to imagine ourselves as the *watchdog* that is looking at the power of the channel in order to see if there is or not the presence of a jammer: if normally the system tends to transmit at 1 mW in the mainband and the attacker transmits random noise at 1.1 mW, no noticeable anomaly will be detected.
If instead the jammer also transmits in the sidebands of the wireless channel, where the device in the network tends to leak a very low amount of power, it will be easy to see the presence of a jammer that transmits at the same power detected in the mainband. In the case the jammer became smart and started injecting noise only in the mainband of the wireless channel, it could be highlighted that no filter in the real world is ideal, and so there will be an increasing amount of power detected in the sidebands of the channel, which according to our system can be meaningful or not.

5 Experiments

5.1 Scenario

The scenario under which the experiments were conducted involves a static configuration, with the following elements present:

- **The cellular base station**, which has to provide a reliable cellular communication between itself and the cellular device.
- **The watchdog**, which has to collect passively data from the wireless channel and determine the presence of a jammer. In addition, the watchdog is intended as a separated element from the base station. The watchdog is supposed to take raw i-q samples (OFDM symbols) from the channel without performing any kind of signal processing (such as channel equalization). Then, in order to collect i-q samples from the channel the watchdog has to be synchronized to the cellular base station, requiring so OFDM-symbol synchronization.
- **The jammer**, which has to inject noise into the wireless channel in order to achieve a denial of service (DoS) in the network.
- **The cellular device**, which has to transmit data whenever it needs to do that.

Here below a scheme of the described scenario:

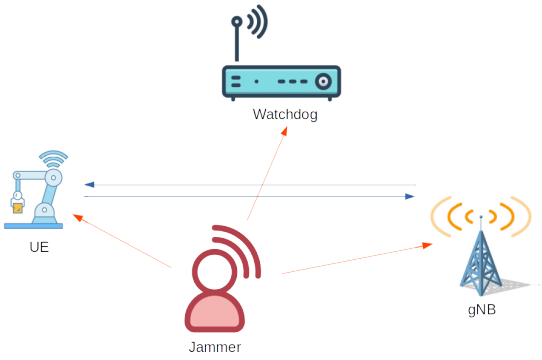


Figure 5.1: Considered security scenario: Blue arrows indicate legitimate cellular communications and red arrows indicate the jamming signals

This kind of described scenario present some advantages:

- **Simplicity**: this approach requires no further digital signal processing functions than OFDM-symbol synchroniza- tion and IQ-sampling. In particular, we neither need equalization, synchronization to the any kind of Physical Resource Block structures, nor decoding or control on data symbols.
- **Generalization:** OFDM is and will remain an important building block in wireless systems, such as 5G-NR, DECT-2020, and WiFi. Our detection model is based solely on data obtained during normal operation and is deliberately not customized to a specific jamming attack, but it relies only on normal behaviour of the system operations.
- **Simple integration:** the watchdog operates as a separate network element. This makes it an easily deployed element in the network. In fact, the data that the watchdog needs to be trained are i-q measurements from the channel in all the operating cases of the cellular device.

In a future commercial deployment, the watchdog will be brought to the cellular network that is willing to watch over; its installation will require only passive traffic inspection, data generation and training. After that, it will be able to operate online in the network by inspecting real time traffic, without any other kind of human supervision or intervention.
Because the data acquisition and training phases do need human supervision, in the next sections there will be described all the methods and details that lead to the realization of the watchdog, explaining then its performances with comments on the obtained results.

5.2 Setup and Data Acquisition

All the roles described in the previous chapter were implemented in the real world as follows:

- The setup is running a private 5G network in the frequency band n78 with cen- ter frequency fc = 2.56 GHz (uplink of the channel) and a 18.43 MHz bandwidth operating in time division duplex (TDD) mode. The base station implements 5G-NR as a software-defined radio (SDR), based on srsRAN 23.5 and on the bladeRF 2.0 micro xA4 radio frequency (RF) frontend .
  The core network functionality is provided by Open5GS 2.6.4, running on the same generic computer as srsRAN. The resulting system operates as a 5G standalone network (without 4G core) and complies with the Release 17.4.0 of the 3GPP Series 38 standards.

- The Cellular device is an unmodified 5G smartphone, namely the Samsung Galaxy A90 (model version: SM-A908B).

- Jammer and watchdog are each implemented as an SDR, based on GNURadio and on the ADALM-PLUTO RF fron- tend (hardware revision B, firmware 0.35).
  The measurements in GNURadio are outputted as a binary file descriptor of i-q samples full of float32s in IQIQIQ order.
  Both devices operate roughly at f-c within 40 MHz bandwidth due to the a sampling rate of 41.44 MHz. Using this bandwidth, the jammer permanently transmits complex noise, as specified below, while the watchdog permanently records i-q samples from the file descriptor written by GNURadio.

- In this lab setup, all devices employ a conventional dipole antenna with approximately 0 dBi gain and are placed approximately 1 m apart. At such close distance, the SNR between the cellular device and the base station is consistently above 30 dB and, thus, not a relevant factor in our experiment. The jammer injects into the channel noise interference with the approximate power of 7 dBm, which is clearly measurable but not high enough to reduce the 5G data rate in this scenario.

- To process data the PC of the lab was using Ubuntu 22.04 LTS as operative system, mounting a SLI of RTX Quadro 5000 as GPU.

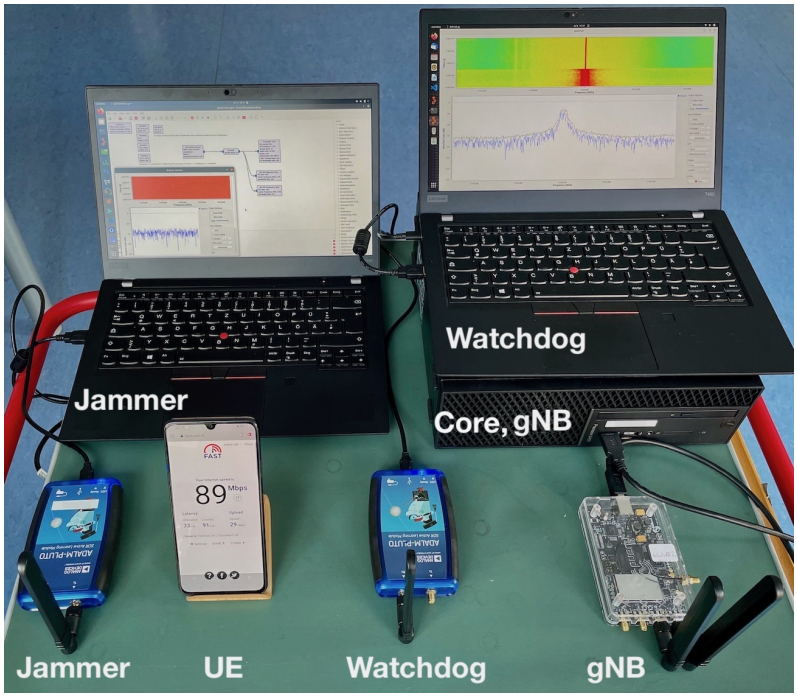


Figure 5.2: Experimental setup: 5G device (indicated as UE) and 3 computers with their respective RF frontends, operating as SDRs for the base station (indicated as gNB), the jammer and the watchdog. The shown distance is for illustration purposes only. During measurements, the antennas were placed 1 m apart.

Data was taken from the three operating situations in which the system can occur:

- Base station ON but the cellular is not transmitting any kind of data.
- Base station ON and the cellular is transmitting data using a speedtest.
- Both of the cases were replicated but with the jammer that is active and transmitting noise.
  The transmitted noise was implemented in two ways: **uniform** noise and **Gaussian** noise.

5.3 Dataset

After the data collection phase, conducted as described above, i-q file descriptors were used to created both data structures, using Python 3.10 programming language.

5.3.1 I-Q Plots

For the i-q plot data structure, the dataset was structrured in the following way:

- First of all, the number n of samples in the plot was decided by prior.
  In this case, n can be: 256, 1024 or 2048.
- The training set was composed by 4000 images: half of them were representing the case in which the cellular device was not transmitting any kind of data, half of them were representing the other case.
- The validation set was composed by 600 images, distributed in the same way as before.
- A set composed only by i-q plots in the jammed case with 500 images was also created.
  This set has not to be considered a validation set because it does not have the same distribution of the training set, but it was used to have more information about performances of the model before conducting the final test.
  For example, after the training phase, it was checked the ratio between the reconstruction error of the validation set and the jammed set. If the reconstruction error was close to 1, it meant that the model was not trained properly, and so conducting other tests was useless.
- The test set is composed by 800 i-q diagrams, containing 400 diagrams in the jammed case that were randomly selected either from the empty or busy subset. The remaining 400 diagrams of the test set are not jammed and, again, equally distributed in the same way of the training and validation set.

Here below few sections of the code developed to implement these tasks:

```
empty_channel = np.fromfile(open("empty_channel"), dtype=np.complex64)
tx_channel=np.fromfile(open("tx_channel"), dtype=np.complex64)
jammer=np.fromfile(open("jammed_channel"), dtype=np.complex64)
## reading file descriptors

training_set_dim=2000
validation_set_dim=300
test_set_dim=200
## setting dimensions
```

```
lower=0
upper=1024 ## 1024 samples for plot
label_string="0_" ## labelling the image
img_index=0
```

```
while(img_index<training_set_dim):
    dataset=empty_channel[lower:upper]
    x = [ele.real for ele in dataset] ## selecting real and imaginary part
    y = [ele.imag for ele in dataset]
    plt.scatter(x, y)
    plt.axis("off")
    plt.savefig("./data/clean/images/"+label_string+ str(img_index)+".png") ##
    plotting and saving
    plt.clf()
    lower+=1024
    upper+=1024
    img_index+=1

while(img_index<training_set_dim+validation_set_dim):
    dataset=empty_channel[lower:upper]
    x = [ele.real for ele in dataset]
    y = [ele.imag for ele in dataset]
    plt.scatter(x, y)
    plt.axis("off")
    plt.savefig("./data/clean_validation/images/"+label_string+ str(img_index)+".png
    ")
    plt.clf()
    lower+=1024
    upper+=1024
    img_index+=1


while(img_index<training_set_dim+validation_set_dim+test_set_dim):
    dataset=empty_channel[lower:upper]
    x = [ele.real for ele in dataset]
    y = [ele.imag for ele in dataset]
    plt.scatter(x, y)
    plt.axis("off")
    plt.savefig("./data/test/images/"+label_string+ str(img_index)+".png")
    plt.clf()
    lower+=1024
    upper+=1024
    img_index+=1
## same procedure as before
```

As it is reported from the code, the plots are saved as RGB images (640x480 format); in order to

save computational complexity and avoid useless information, the images were at first converted to the grayscale colorspace (the color of the dots in the plot has no meaning for this purpose) and then downsampled to a 128x128 format.
The images were also labeled for each case in the following way:

- 0 for the empty channel case
- 1 for the transmitting channel case
- 2 for the jammed channel case

Also, at each image was assigned a progressive number, in order to have unique identifiers. It is then useful to report that during this plotting phase it was also possible to fix the axis size; this information will become useful later.
Some examples of the plots created using this code are reported in Figures 5.3 and 5.4:

Figure 5.3: Plots in the no transmission case with n=256: on the left the plot with axis normalization, while on the right the plot with the axis range fixed to [-1.5;1.5]

Figure 5.4: Other examples of plots for n=256 case: the first one represents the transmitting channel case, the second one an uniform jammed channel and the third one a Gaussian jammed channel.

The data plotted has shown then that the system in the transmission case is using a 4-QAM constellation scheme. It is easy to see how the most majority of samples are concentrated into the four corners of a rectangle.

5.3.2 Waterfall Plots

Regarding the waterfall plot data structure, the number of data structures contained in the respective training, validation and test set was the same as described before, due to the fact that in both cases a CAE was adopted to perform data analysis.
It is then useful here describe how waterfall plots were structured and stored:

- The PSD array was computed as follows: $PSD = |fft(x)|^2/(n \times sampling\_rate)$.[12] Where x was an array containing n i-q samples. This then leads to an array with length n; in this case, n was set to 1024.

- The waterfall plots were composed by 50 stacked PSD arrays, in order to create a 50×1024 matrix.

- As it was said before, due to the high computational complexity of representing the plots as images, the images were stored as numpy matrices, so no graphical representation is available here.

To make the reader better understand the different behavior of the channel in different states in terms of power, examples of PSD (for $n = 1024$) in the four different states that were considered will be shown below.

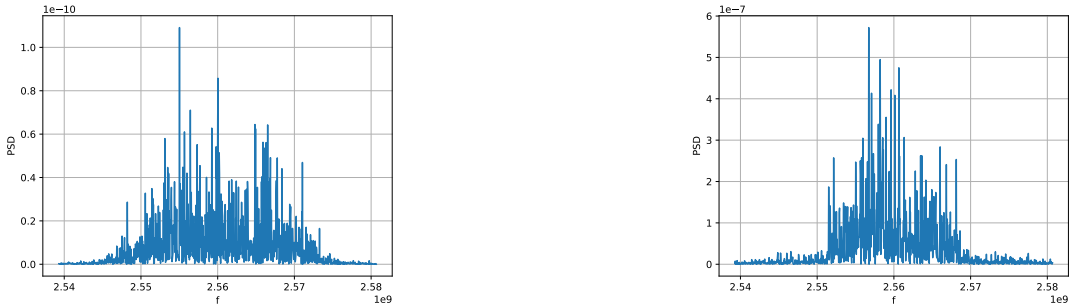


Figure 5.5: On the left, a plotted PSD in the case: "empty channel"; on the right, a plotted PSD in the case: "transmitting channel".

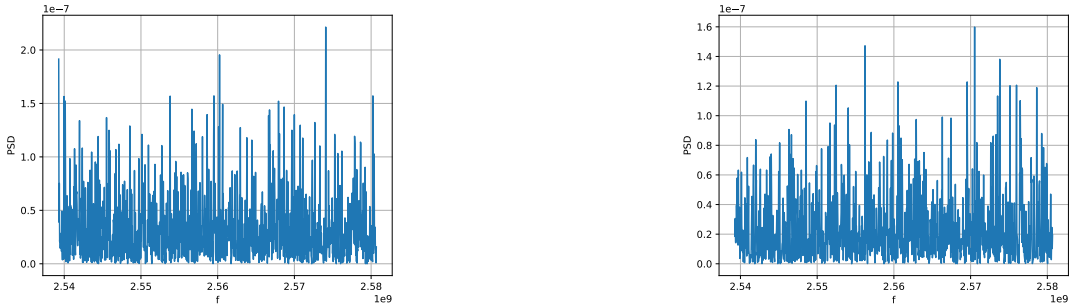


Figure 5.6: On the left, a plotted PSD in the case: "uniform jammed channel"; on the right, a plotted PSD in the case: "Gaussian jammed channel".

As can be seen from fig 5.5 and 5.6, the empty channel is easily distinguished because of the low levels of spectral density present on all frequencies; the transmitting channel, on the other hand, can be distinguished by looking at the sidebands, as it tends to leak little energy compared to that being transmitted in the mainband.
Here below a section of the code developed to implement this task:

```
wat_size=50
num_waterfall_train=2000
num_waterfall_val=300
num_waterfall_test=200 ## setting the parameters
n=1024
index=0
lower=0
upper=lower+n
label="0"
for i in range(num_waterfall_train): ## creating and saving the waterfall plots as
    numpy matrices
    mat=np.zeros((wat_size,n)) ## empty matrix to fill
    for j in range(wat_size):
        x=empty_ch[lower:upper]
        PSD = np.abs(np.fft.fft(x))**2 / (n*samp_rate) ## PSD computation
        PSD_shifted = np.fft.fftshift(PSD) ## centering around fc
        for p in range(n):
            mat[j:]=PSD_shifted
        lower+=n
        upper+=n
    np.save("matrices/clean/"+label+str(index),mat)
    index+=1
```

Also here matrix were labeled for each case with a unique ID as before.

5.4 Data Analysis

5.4.1 I-Q Plots

After having created the dataset as described above, the next task was to create and train a ML model that was capable of distinguishing an i-q plot taken from a channel in a trusted state and from a jammed one; here below a table that shows the layers adopted to build the model: The model, so can be summarized in as follows:

- The model takes as input a $128 \times 128 \times 1$ data structure, and so a grayscale 128×128 image.
- The first 2 layers are two classical convolutional layers with stride equal to two in order to reduce the size of the input data.

Table 5.1: Structure of the employed CAE

	Layer	Output size	No. of parameters
Encoder	Input	$128 \times 128 \times 1$	0
	Convolutional 1	$64 \times 64 \times 64$	640
	Convolutional 2	$32 \times 32 \times 32$	18464
	Flatten	32768	0
	Dense	32	1048608
Decoder	Input	32	0
	Dense	32768	1081344
	Reshape	$32 \times 32 \times 32$	0
	Convolutional 1^T	$64 \times 64 \times 32$	9248
	Convolutional 2^T	$128 \times 128 \times 64$	18496
	Convolutional	$128 \times 128 \times 1$	577

- Then data is *flattened* in order to be connected to a fully connected layer composed by 32 neurons.
  Here the encoder ends.
- The decoder adds then a fully connected layer with the same number of neurons of the encoder; the data is then reshaped to a $32 \times 32 \times 32$ structure in order to be processed with convolutional layers.
- The next two convolutional layers have the same size of the previous one but with the addition of upsampling operations in order to get back to the original input dimensionality.
- At the end, it is applied a classical convolutional layer to have as output dimensionality $128 \times 128 \times 1$.
- The chosen loss for the training of the model was the MSE between the input data and the output data.
- To build the model it was used Keras API.[13]
- The model, built as described, weights approximately 6 MB.
- The number of trainable parameters of the model is 2177377.
- The model was tuned for the n=1024 case, then tests were performed for other time resolutions.

```
## define our neural network
class AutoEncoder(Model):
    def __init__(self):
        super(AutoEncoder, self).__init__()
        self.encoder = Sequential([
            Conv2D(64, 3, strides=2, padding="same", activation="sigmoid",
    input_shape=(128,128,1)),
            Conv2D(32, 3, strides=2, padding="same", activation="sigmoid"),
            Flatten(),
            Dense(32, activation = "sigmoid")
        ])
        self.decoder = Sequential([
            Dense(32*32*32, activation="sigmoid",input_shape=self.encoder.output.
    shape[1:]),
            Reshape((32,32,32)),
            Conv2DTranspose(32, 3, strides=2, padding="same", activation="sigmoid"),
            Conv2DTranspose(64, 3, strides=2, padding="same", activation="sigmoid"),
            Conv2D(1, 3, strides=1, padding="same", activation="sigmoid")
        ])
    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

Before being processed into the model, data and its labels were loaded directly into the memory of the machine, downsampled to the correct scalespace and then converted into grayscale colorspace using OpenCV library implemented in Python.[14]
Also, the intensities of the pixels were normalized before the data was processed into the model. This kind of data processing allows the model to not distort the structure of the data while shrinking the value range of the data, in order to give less importance to any outliers.
Because the range of the values of data was shrinked to $[0;1]$, the sigmoid activation function was used in all the layers of the model (recall that the output of sigmoid is in the range $[0;1]$).

$$f(x) = \frac{1}{1+e^{-x}}$$

Figure 5.7: Sigmoid activation function

```
clean = []
labels_clean =[]
```

```
val = []
labels_val =[]
anomalies = []
labels_anomalies =[]
test=[]
labels_test=[]
## empty arrays that will contain the images of the four datasets and the labels of
    the corresponding images; in this case 0 is a clean channel and 1 is a jammed
    channel
for folder in tqdm(os.listdir("data_Gaussian_1024")):
  print(folder)
  for im in tqdm(os.listdir("data_Gaussian_1024/"+folder+"/images/")):
      if folder == "clean":
          clean.append(cv.cvtColor(cv.resize(cv.imread("data_Gaussian_1024/"+folder+
    "/images/"+im),(128,128)),cv.COLOR_BGR2GRAY).reshape(128,128,1))
          labels_clean.append(0)
      elif folder == "clean_validation":
          val.append(cv.cvtColor(cv.resize(cv.imread("data_Gaussian_1024/"+folder+"/
    images/"+im),(128,128)),cv.COLOR_BGR2GRAY).reshape(128,128,1))
          labels_val.append(0)
      elif folder== "jammed_channel":
          anomalies.append(cv.cvtColor(cv.resize(cv.imread("data_Gaussian_1024/"+
    folder+"/images/"+im),(128,128)),cv.COLOR_BGR2GRAY).reshape(128,128,1))
          labels_anomalies.append(1)
      else:
        test.append(cv.cvtColor(cv.resize(cv.imread("data_Gaussian_1024/"+folder+"/
    images/"+im),(128,128)),cv.COLOR_BGR2GRAY).reshape(128,128,1))
          labels_test.append(mapping[int(im[0])])
  ## the images were loaded and put in the lists, but was performed also the resize
    of the image to 128x128 and conversion of the colorspace from RGB to grayscale.
```

For each case, the loss of the model saturated at 20 epochs.
Due to the fact that each i-q plot contains a different number of samples, it is expected that the value of the loss of the model trained with different datasets will be different.

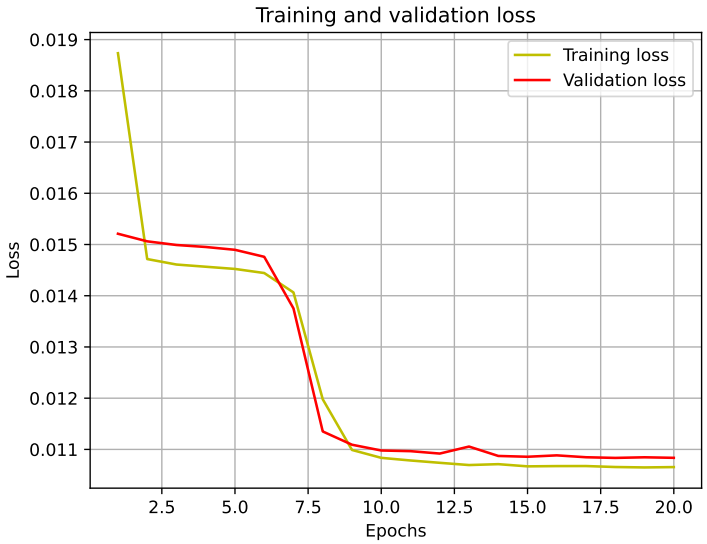


Figure 5.8: Training and validation loss for the case: Gaussian 1024.

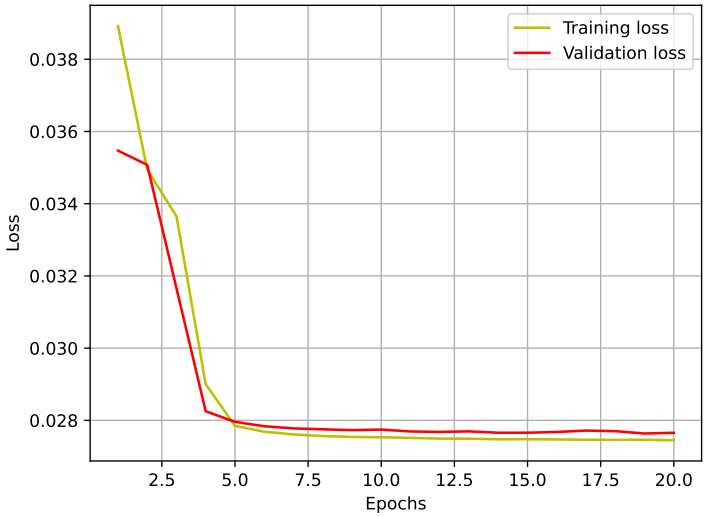


Figure 5.9: Training and validation loss for the case: Uniform 1024.

From these two figures we can see that even if the training set is the same (n=1024 in both cases), the loss is different: this was given by the fact that without taking into the account the amplitude of the i-q samples (and so not fixing the axis sizes) was producing poor results in the

test set when dealing with Gaussian noise.
The problem was then solved by fixing the axis range of the plots to a range that allowed to view all the constellation(determined using a small i-q file descriptor), instead of making the *matplotlib* library take this decision. More details will be given in the result section.
After the training of the model, it was useful to plot also the reconstructed image of the model, in order to see what the model understood from the training set.
Even if it was done to research purposes, it has to be clear that the goal of this model is not to be a good image reconstructor, but a good discriminator between trusted and jammed case. So, even if the reconstruction is not graphically good for a human, it could lead to a watchdog with good performances.
For sure a model with more weights would lead to a better reconstruction, but here it is important also to take into account that the model is also studied to be implemented in a future as an online watchdog, so making it lighter can lead to faster responses to the channel.

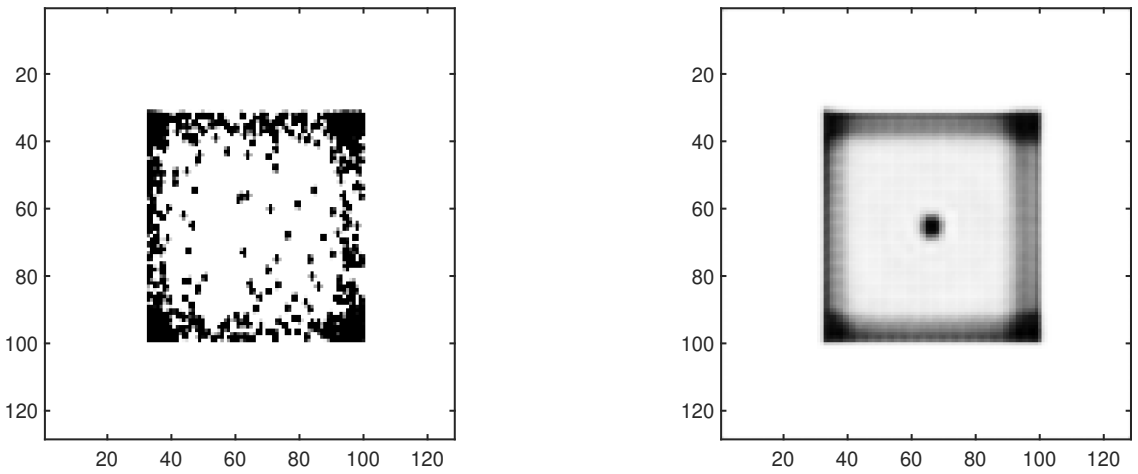


Figure 5.10: Example of 1024 unequalized 4-QAM constellations and their reconstruction by the CAE. The axes labels index the data points in $\mathbb{C}$.

From the image shown above, it is clear that the model has learnt well that the channel can be either in the no transmission case (a lot of samples nearby the origin of the axis), either in the transmission case (4-QAM scheme).

5.4.2 Waterfall Plots

As it was said before, waterfall plots were computed as 50×1024 matrices, and the model used to process data have a similar structure to the one used for the i-q plots processing. Here below

a table of the layers used to build the model:

Table 5.2: Structure of the employed CAE

	Layer	Output size	No. of parameters
Encoder	Input	50×10241	0
	Convolutional 1	$24 \times 511 \times 16$	160
	Convolutional 2	$11 \times 255 \times 32$	4640
	Convolutional 3	$5 \times 127 \times 64$	18496
	Convolutional 4	$2 \times 63 \times 128$	73856
	Flatten	16128	0
	Dense	32	516128
Decoder	Input	32	0
	Dense	532224	1081344
	Reshape	$2 \times 63 \times 128$	0
	Convolutional 1^T	$5 \times 127 \times 64$	73792
	Convolutional 2^T	$11 \times 255 \times 32$	18464
	Convolutional 3^T	$23 \times 511 \times 16$	4264
	Zero padding	$25 \times 512 \times 16$	0
	Convolutional 4^T	$50 \times 1024 \times 1$	145

The total number of trainable parameters of the model is then 1242529.
The only new element with respect to the previous model is the **zero padding** layer: this layers allows to add some rows or columns in a matrix (filled with zeros) in order to modify the dimensionality of a data structure.
During the training of the model, two main problems arose:

- The model needed too many epochs to reach the optimal training. Then, it was placed a monitor on the validation loss with a patience of 7.

- The matrices, as they were created, were not reaching the optimal training even after more than 2000 epochs.
  This problem was solved by applying a monotonic function on the values of the matrices in order to have no more values in the order of 10^{-14}, which create issues with the gradient descent optimization method. The monotonic function was: $f(x) = -log(x)$.

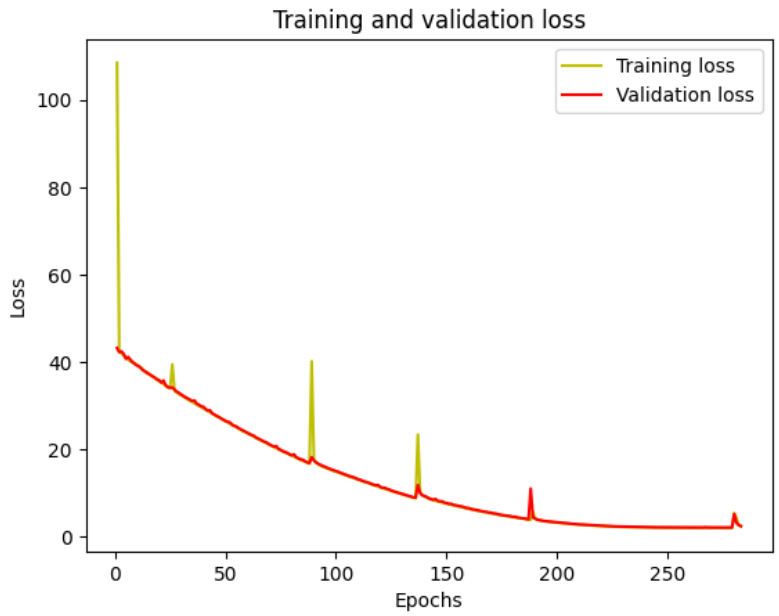


Figure 5.11: Training and validation loss for the waterfall plots.

5.5 Results

5.5.1 I-Q Plots

After the training phase described as above, the model was tested on 6 different test sets:

- n=256, noise=Uniform
- n=1024, noise=Uniform
- n=2048, noise=Uniform
- n=256, noise=Gaussian
- n=1024, noise=Gaussian
- n=2048, noise=Gaussian

While the distribution of the test set is described in the previous section, it is important to underline that for each test set it was created a plot which shows the probability of **false alarm** (false positive) and **missdetection** (false negative) for a varying threshold of the reconstruction error.

The range of the threshold was computed by taking the clean validation set and the jammed dataset and selecting the minimum and maximum reconstruction error computed on both of them.

```
## functions that computes the mean, stdv, min and max of the reconstruction error
def calc_recon_error(batch_images):

    recon_error_list=[]
    for im in trange(0, batch_images.shape[0]):

        img  = batch_images[im]
        img= img.reshape(1,128,128,1)
        recon_error_list.append(model.evaluate(img,img))



    average_recon_error = np.mean(np.array(recon_error_list))
    stdev_recon_error = np.std(np.array(recon_error_list))
    maximum= max(recon_error_list)
    minimum= min(recon_error_list)

    return average_recon_error, stdev_recon_error, maximum, minimum
```

Letting $\mathcal{H}_0$ be the hypothesis of no jamming and $\mathcal{H}_1$ the hypothesis of jamming, the detection of jamming is performed by the following *test function* on the input image X to obtain decision $\hat{\mathcal{H}}$:

$$\hat{\mathcal{H}} = \begin{cases} \mathcal{H}_0; & \Gamma < \tau, \\ \mathcal{H}_1; & \Gamma \geq \tau, \end{cases} \tag{5.1}$$

where τ is a suitable threshold. The two performance met- rics are then mathematically defined as follows:

$$P_{\text{FA,C}} = \mathbb{P}[\hat{\mathcal{H}} = \mathcal{H}_1 | \mathcal{H} = \mathcal{H}_0] \tag{5.2}$$

$$P_{\text{MD,C}} = \mathbb{P}[\hat{\mathcal{H}} = \mathcal{H}_0 | \mathcal{H} = \mathcal{H}_1], \tag{5.3}$$

where $\mathcal{H}$ is the true condition (jamming or no jamming) when observing X. In general, reducing one probability increases the other.
Because of the distribution of the dataset (50% non jammed and 50% jammed) and how the prediction is performed, it is clear then that in the worst case the metrics can reach at maximum the value of 0.5. Here below there will be shown the performances for each test set:

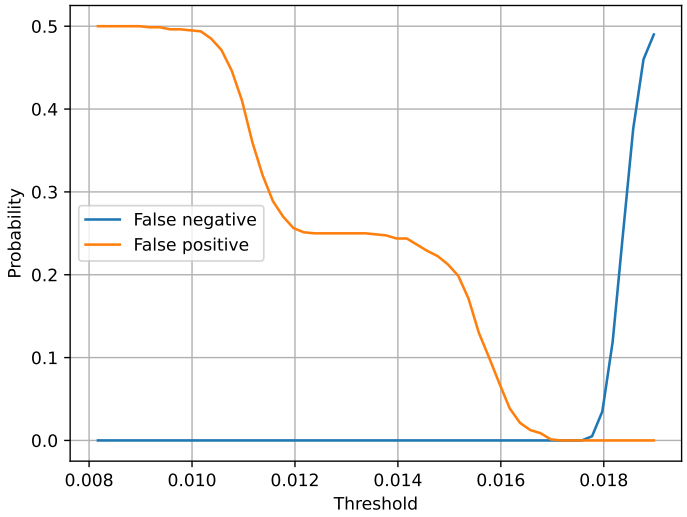


Figure 5.12: Plotted performances for the case: uniform 256.

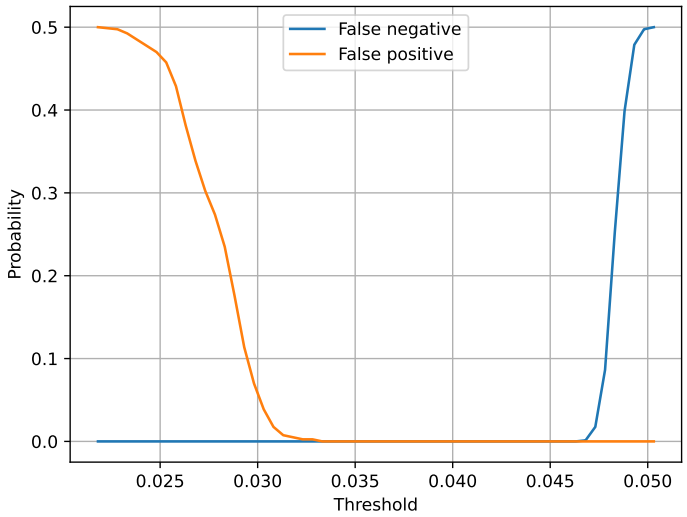


Figure 5.13: Plotted performances for the case: uniform 1024.

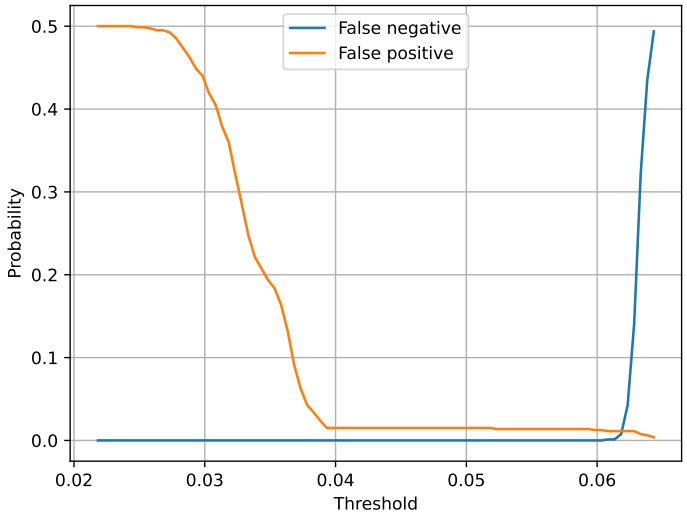


Figure 5.14: Plotted performances for the case: uniform 2048.

Managing Gaussian noise required more attention: in fact, without fixing the axis sizes, the Gaussian jammed channel looks similar to a empty channel (see figures above). After fixing the axis sizes, the performances were significantly better.

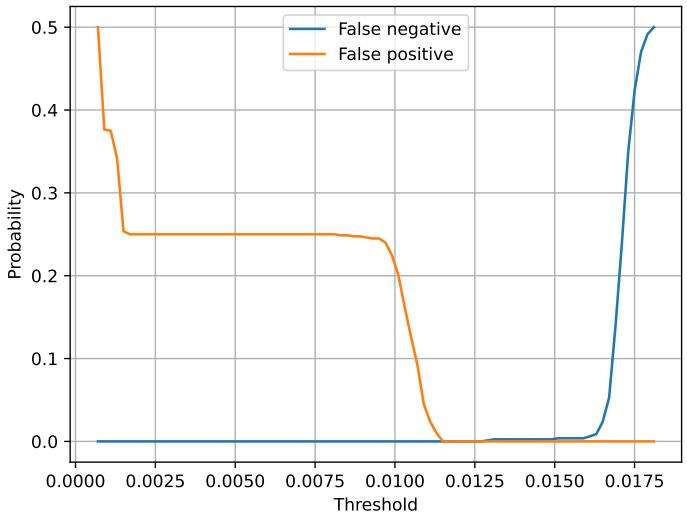


Figure 5.15: Plotted performances for the case: Gaussian 256 (fixed axis sizes).

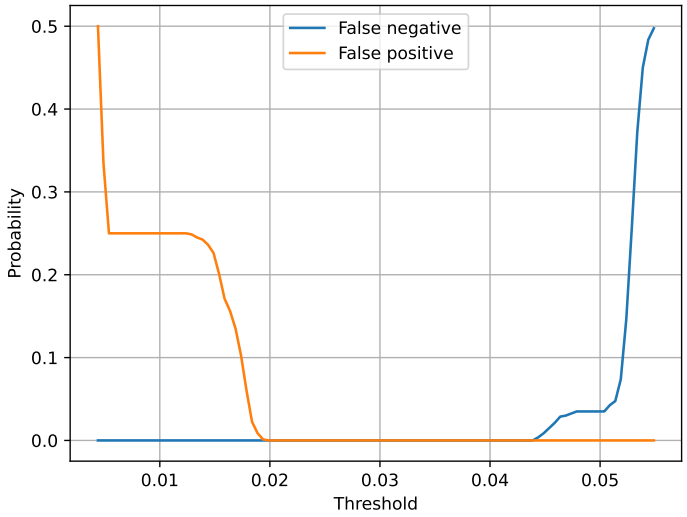


Figure 5.16: Plotted performances for the case: Gaussian 1024 (fixed axis sizes).

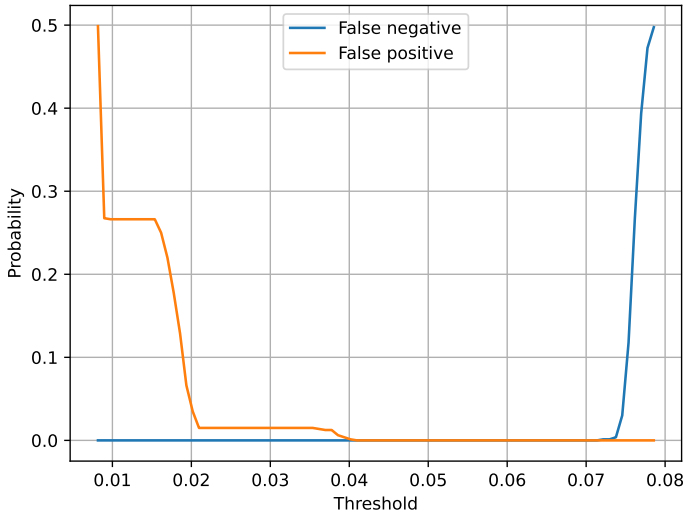


Figure 5.17: Plotted performances for the case: Gaussian 2048 (fixed axis sizes).

After having computed the plots for the false alarm and missdetection rates, it was chosen also to simulate the behaviour of the watchdog: on the watchdog it was set a threshold for performing anomaly detection ($\mu + \sigma$ of the reconstruction errors computed in the validation set) and plotted confusion matrices for each case.

Table 5.3: Confusion Matrix for the case (Gaussian, 1024), Overall accuracy=91%

		Predicted	
		Clean	Jammed
True	Clean	334	66
	Jammed	0	400

Table 5.4: Confusion Matrix for the case (Uniform, 1024), Overall accuracy=96%

		Predicted	
		Clean	Jammed
True	Clean	372	28
	Jammed	0	400

Table 5.5: Confusion Matrix for the case (Gaussian, 256), Overall accuracy=88%

		Predicted	
		Clean	Jammed
True	Clean	309	91
	Jammed	0	400

Table 5.6: Confusion Matrix for the case (Uniform, 256), Overall accuracy=90%

		Predicted	
		Clean	Jammed
True	Clean	322	78
	Jammed	0	400

Table 5.7: Confusion Matrix for the case (Gaussian, 2048), Overall accuracy=90%

		Predicted	
		Clean	Jammed
True	Clean	327	73
	Jammed	0	400

Table 5.8: Confusion Matrix for the case (Uniform, 2048), Overall accuracy=95%

		Predicted	
		Clean	Jammed
True	Clean	360	40
	Jammed	0	400

After having exposed these results, comments on them will be written in the next chapter.

5.5.2 Waterfall Plots

After having trained the model as explained before, some tests were conducted on the pixels of the arrays with the aim of highlighting the parts of the spectrum with the highest reconstruction error.
This approach then follows the same assumptions and methodologies used for image segmentation with autoencoders.[15]
In particular, it was assumed that pixels with higher reconstruction error represent parts of the spectrum subject to the action of the jammer, due to the fact that the model has learnt to reconstruct only an empty channel (so with low power values present in every part of the spectrum) or the channel with a transmission ongoing (so with high power values in the mainband of the channel).
Due to the reasons explained above, it is expected that in a fully jammed spectrum the parts with higher reconstruction error will be the sides of the spectrum, in which the model does not expect power values in the order of magnitude of the ones used with an ongoing transmission in the mainband.

It was decided so to set a **heatmap** on the reconstruction error of the cells of the matrices and with that see if the model shows the expected results. Lower values in the heatmap are represented with blue, while higher values are represented in red.
The scale of the heatmap was created by setting as minimum 0 and maximum the maximum reconstruction error of a pixel found in the additional jammed set. Values higher than this threshold will be clipped to the maximum in the heatmap.

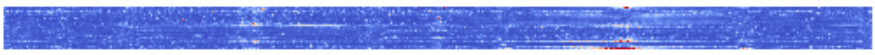


Figure 5.18: Heatmap of the reconstruction error of a waterfall plot in the transmission case.

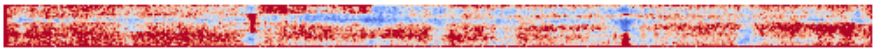


Figure 5.19: Heatmap of the reconstruction error of an uniform jammed waterfall plot.

Figures 5.17 and 5.18 tell two main things:

- The model has learnt well how to reconstruct a waterfall plot: in fact, in the clean case, most of the waterfall plot is painted in blue.
- As expected, the parts with higher reconstruction error are located in the sides of the plots.

All the other heatmaps computed followed the same structure presented earlier.
The model, so, has confirmed that the presence of the jammer has to be determined by looking at the sidebands of the channel.
It was decided then to compute statistics about the energy in the sideband of the channel.

- First of all, the sidebands of the channel have benn set as 1 MHz on the left and on the right of the mainband, making them almost 10% of the mainband of the channel.(remember to cite the article)
- To compute the statistics, for each case 200 waterfall plots were analyzed.
- Saving computational complexity in this case was not so critical as before, so to collect more data waterfall plots were composed by 100 stacked PSDs, makin them 100×1024 matrixes.
  The PSDs that compose the waterfall plot are computed in the same way as before.

- The first quantity that it was decided to plot was the energy (energy) in the sidebands among a waterfall plot window.

- The second was the ratio between the energy in the sidebands and the total energy in the channel (basically mainband + sidebands).

- For those quantities, it was plotted the CDF in each possible case of the channel. Actually, it was also tried to plot the PDF of those values with Kernel Density Estimation, but due to the high variability of data the results were poor.

Here below all the plots computed:

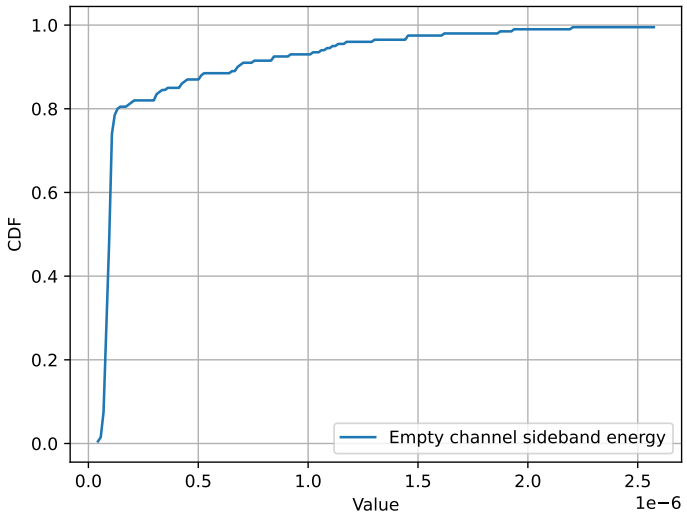


Figure 5.20: CDF of the energy in the sidebands in the case: empty channel.

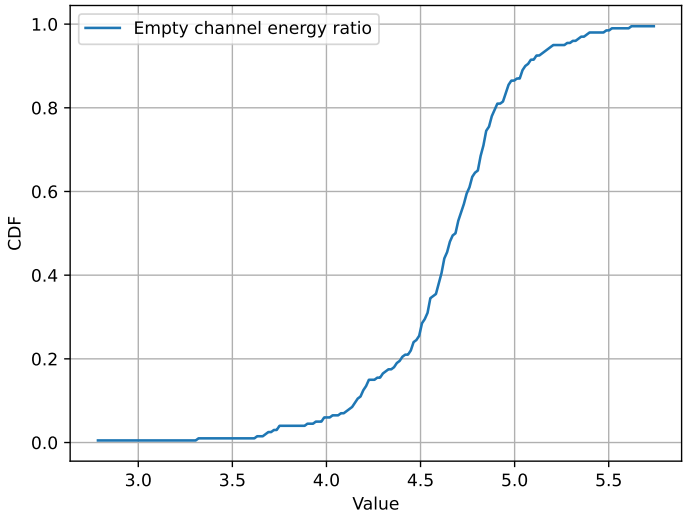


Figure 5.21: CDF of the ratio (in %) in the sidebands in the case: empty channel.

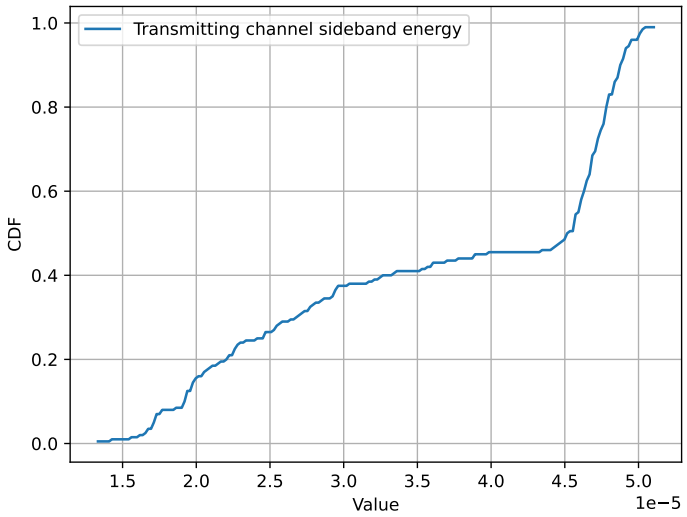


Figure 5.22: CDF of the energy in the sidebands in the case: transmitting channel.

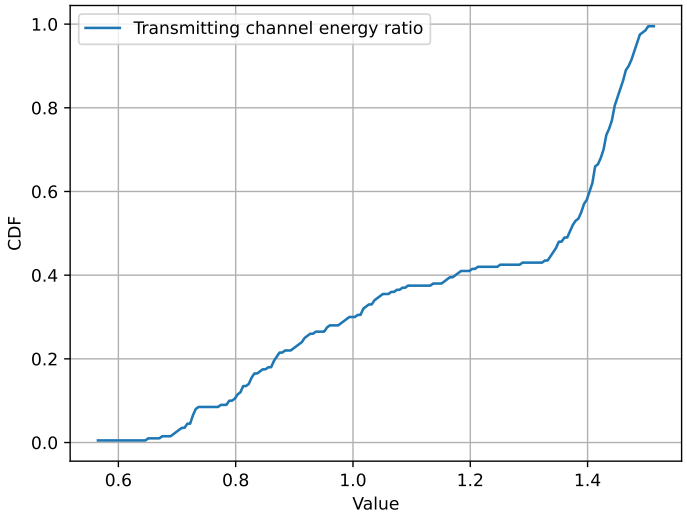


Figure 5.23: CDF of the ratio (in %) in the sidebands in the case: transmitting channel.

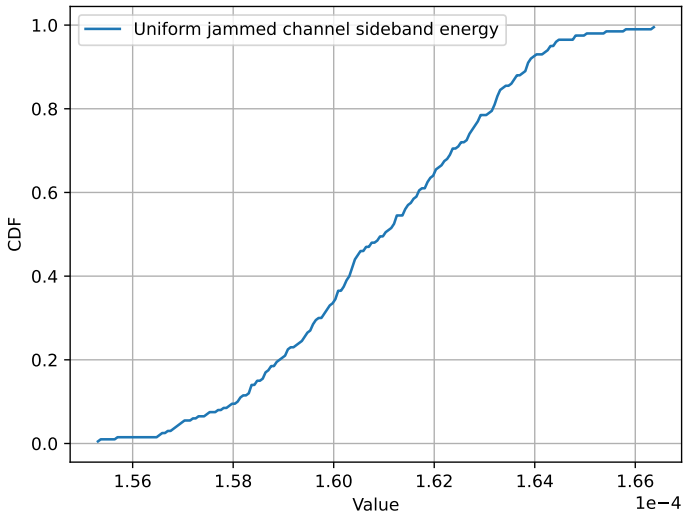


Figure 5.24: CDF of the energy in the sidebands in the case: uniform jammed channel.

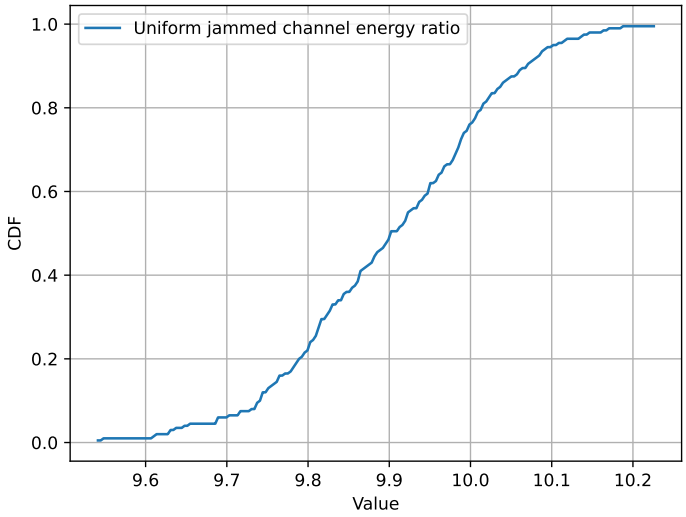


Figure 5.25: CDF of the ratio (in %) in the sidebands in the case: uniform jammed channel.

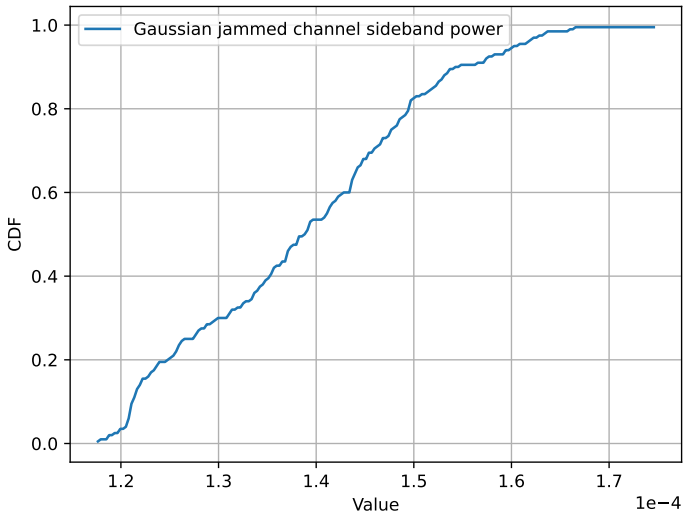


Figure 5.26: CDF of the energy in the sidebands in the case: Gaussian jammed channel.

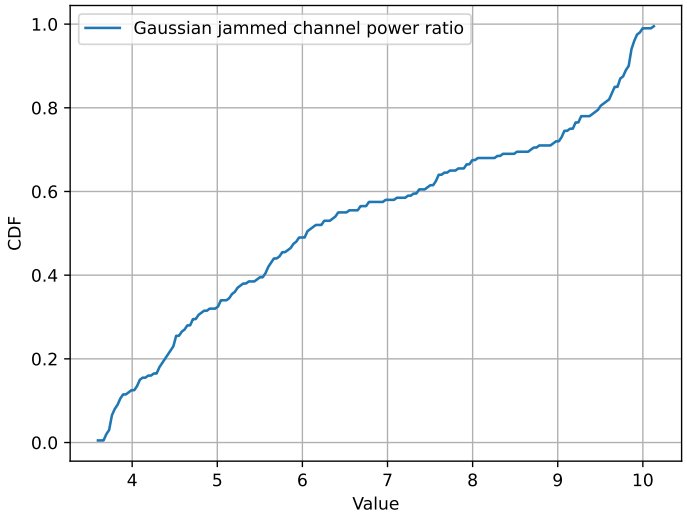


Figure 5.27: CDF of the ratio (in %) in the sidebands in the case: Gaussian jammed channel.

Also for this case, the comments on these results will be done in the next chapter.

6

Conclusion and future works

In this thesis it has been proposed a preliminary study on how to detect jamming attacks against cellular systems, with a focus on 5G networks. This technique only exploits the structure of physical-layer signals with a loose attachment to the network.
While our watch- dog still has to synchronize with the RAN at the OFDM- symbol level, it requires no estimation, no equalization, and no decoding. This loose design is agnostic to higher-layer structures, such as headers and frames, and can be implemented separately from the operator network and the user device. This is a substantial simplification and generalization, compared to conventional intrusion detection systems.
In essence, the work previously described takes two approaches to detect or not detect the presence of a jammer:

- The first approach consists on applying a CAE on the i-q plots of a 5G testbed, with provided first experimental results in the high-SNR regime. These preliminary results show that the proposed detector is able to perfectly distinguish between the attack and no attack conditions, thus, demonstrating to be a promising security mechanism.
- The second approach is to enter the frequency domain of the channel, and in particular to study how the spectral power of the channel varies with respect to the channel states studied. For reasons previously described, the investigation was carried out on the sidebands of the channel, assuming for true the fact that, each filter being non perfect, the presence of a jammer would in any case lead to a leak in the sidebands considered, no matter how much it attempts to inject power into the spectrum in which the system operates.

This approach, however, shows its limitations as a jammer becomes increasingly accurate and begins to leak little power into the channel sidebands, making itself very much like a trusted state. For this reason, any future research is likely to focus on the first method presented, which uses more sophisticated solutions and can identify jammers regardless of the amount of power they leak into the channel sidebands.

Having done this preliminary study, it is therefore appropriate to make some considerations about the possible directions this research might take: the most likely direction to be taken in the future will be to move from the preliminary study to a practical approach, as all studies on the performance of the adopted models have been carried out offline.
A similar pattern will then be adopted:

- The CAE adopted for jammer detection will actually be run on a machine, which will take care of providing it with all the necessary computing power.

- It will be precisely framed the scenario in which the watchdog will operate: on this, constellation measurements will be taken concerning all channel states deemed *trusted*. Once the data useful for training is collected, the model will be trained on it as shown above. Finally, a threshold deemed optimal for the approached problem will be established; the criteria for establishing it are exactly those used previously: based on a validation set, the statistics of it regarding the loss function will be extracted and on these calculated the threshold analytically. It is therefore implied that since the training is jammer-independent, even the threshold calculation will only have to take into account the trusted states of the channel with which the model was trained.
  Previously, the threshold had been set as: $\mu + \sigma$ of the reconstruction errors computed in the validation set leading to good results, but also $\mu + 2\sigma$ could be an optimal solution depending on the problem it is willing to be approached. In general, a lower threshold leads to a pessimistic watchdog (with higher FA rates), while an higher one leads to an optimistic threshold (with higher MD rates).

- A time window (fft size) will then be established with which the watchdog will collect information from the channel to create the i-q diagram. To save computational capacity, it is possible that the watchdog will not constantly monitor the channel, but will do so at set intervals (e.g.: 5 seconds). Once the jammer is found, it could scroll back through the memory of the descriptor file containing the i-q coordinates to figure out the precise moment when the channel started to be in a corrupted state.
  Using the latter information and multiple watchdogs (three or more), it is possible to combine the information regarding the instant when each individual watchdog launches an alarm to locate the jammer in a 3D space.

After these remarks, it is worth mentioning (before ending this thesis) that on this work it has been written a scientific paper titled *Detecting* 5*G Signal Jammers with Autoencoders Based on Loose Observations*, written by Matteo Varotto, Stefano Tomasin and Stefan Valentin.

References

[1] https://www.marketsandmarkets.com/Market-Reports/iot-manufacturing-market-129197408.html.

[2] https://www.statista.com/statistics/666864/iot-spending-by-vertical-worldwide/.

[3] https://www.cybersecurity360.it/nuove-minacce/ransomware/toyota-sospende-lattivita-per-un-attacco-informatico-perche-e-evento-notevole/.

[4] https://arxiv.org/abs/1511.04352.

[5] https://www.ling.upenn.edu/courses/cogs501/Gallant1990.pdf.

[6] https://web.mit.edu/course/other/i2course/www/vision_and_learning/perceptron_notes.pdf.

[7] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=708428.

[8] https://www.sciencedirect.com/science/article/pii/S0950705121009576.

[9] https://pubs.aip.org/aip/rsi/article/65/6/1803/682910/Neural-networks-and-their-applicationsNeural.

[10] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9451544.

[11] https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[12] https://pysdr.org/content/frequency_domain.html.

[13] https://keras.io/api/.

[14] https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html.

[15] https://ieeexplore.ieee.org/document/7952482.

Acknowledgments

A Zlatan Ibrahimović
Ad Enzo Conte