Wireless Jamming Early Detection and Classification Using Deep Learning

A Graduation Project Report Submitted to the faculty of Engineering at Cairo University in Partial fulfilment of The Requirements for the Degree of Bachelor Engineering

In

Electronics and Electrical Communications Engineering

Team Members:

Reem Abdelfattah Abdelaziz

Omnia Mostafa Mohamed Ali

Mohamed Atef Rabie Hussein

Nancy Khaled Farouk Abdelhamid

Nesma Gaber Ibrahim Kandel

Under supervision of:

Dr. Hassan Mostafa

Faculty of Engineering, Cairo University

Giza Egypt

July 2023

This Page is Intentionally left empty.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

MIMO	Multiple-Input Multiple-Output
Mm-Wave	Millimeter-Wave
NOMA	Non-Orthogonal Multiple Access
IOT	Internet of Things
WSNs	Wireless Sensor Networks
RPAS	Remotely-Piloted Aircraft System
DoS	Denial of Service
RF	Radio Frequency
SNR	Signal to noise ratio
BER	Bit error rate
BPSK	Binary phase shift keying
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-term Memory

Acknowledgments

Firstly, we thank God for His abundant provision, constant grace, and never-ending blessings. All goodness comes from Him alone, glory be to Him. We are grateful to God that He has enabled us to complete our educational journey and reach this point. We thank Him for granting us this wonderful opportunity to work on this project, benefit from it, and hopefully make it beneficial to others. We thank God that we have reached where we are now, despite the many challenges we have faced, which were overcome through His facilitation and grace. We cannot enumerate enough praise to God for completing His blessings and assistance upon us, and we acknowledge that without God's guidance, we have no ability or power.

{ مَّآ أَصَابَكَ مِنۡ حَسَنَةٖ فَمِنَ ٱللَّهِۖ } سُورَةُ النِّسَاءِ: ٧٩

Secondly, we would like to thank our supervisor who was very dedicated to ensuring that we perform to the best of our abilities and that our work is of the highest standard and quality. We are proud to have worked under his supervision. Dr. Hassan Mostafa

Thirdly, we would like to thank those who have helped and supported us throughout the completion of this task. They have been a great assistance to us and have not spared anything, whether it be knowledge, time, or effort. Dr. Hassan Abushady and Eng. Abdurahman Emad.

Lastly, we would like to thank the company SeamlessWaves semiconductors for hosting us during the project and providing us with all the resources and support we needed to complete our work successfully. They were very helpful, patient, and kind towards us.

قال النبيُّ صلَّى اللهُ عليه وسلَّم "من لم يشكرِ النَّاسَ لم يشكرِ اللَّهَ"

Abstract

Wireless network security has become increasingly demanding, particularly with the widespread adoption of wireless technologies such as IoT devices. This project focuses on the early detection and classification of anomalies in I/Q samples, specifically targeting jamming attacks.

To achieve this, we initially utilized a publicly available real dataset from USRP as training data for our deep learning models. Subsequently, we created our own dataset, consisting of three classes of jamming: No jamming (silent), Gaussian jamming, and Tone jamming. The dataset also incorporated varying levels of jamming power to emulate real-world scenarios. Deep learning algorithms were developed to effectively detect and classify jamming attacks.

In our jamming detection model, we employed both Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) models, comparing their performance. Various architectural configurations were explored for each model to optimize their accuracy and efficiency. Impressively, our models achieved an average accuracy of 99% across different Signal-to-Noise Ratio (SNR) levels and 99% for varying jamming power values on our generated dataset.

To further enhance the applicability of our models, we implemented post-training quantization techniques to reduce memory consumption and accelerate computations. We anticipate testing the performance of our models, both pre and post quantization, using the BladeRF x40 board in baseband mode.

Chapter 1: Introduction

With the rapid proliferation of wireless devices, the explosion of Internet-based mobile applications under the driving forces of artificial intelligence and IOT devices, wireless services have penetrated every aspect of our lives and become increasingly important as an essential component of the telecommunications infrastructure in our society. In the past two decades, we have witnessed the significant advancement of wireless communication and networking technologies. However, these innovative wireless digital communication systems like 5th generation (5G) and Wi-Fi face an important problem, which is wireless jamming.

Wireless jamming poses a significant threat to digital communication systems, jeopardizing their integrity, reliability, and security. In an increasingly connected world, where wireless communication is the backbone of numerous applications and services, understanding the concept of wireless jamming and its potential dangers is of utmost importance. [1]

1.1. Jamming Definition

Jamming in wireless networks is defined as the intentional disruption of existing wireless communication by decreasing the signal-to-noise ratio SNR at the receiver side through the transmission of wireless interfering signals of relatively high power. There are many types of these wireless interfering signals such as, gaussian jamming, tone jamming, multi-tone jamming, sweep jamming, barrage jamming … etc. we will focus on gaussian, and tone jamming types and illustrate them in the following sections.

Jamming attacks mainly target the physical layer but can also target higher layers of the wireless networks. Under these attacks, the desired communication between a transmitter at location A and a receiver at location B is interrupted by jamming signals which keep the channel busy. When the jamming signals occupy the channel for a longer period of time, they can create a denial of service (DoS). [2]

1.2. Importance of Jamming detection

Detecting Anomalous signals like jamming attacks is important because it is the first step towards building a secure and dependable wireless network. By simply injecting high-power signals into the wireless communication channel, an adversary can disrupt any wireless links, preventing all Radio Frequency (RF) communications to occur in a specific geographical area. It can also be done at a relevantly low cost. Defense against jamming attacks has been an increasing concern for the military and disaster response authorities. The military uses jamming attacks as a tool to attack and disrupt terrorist's communications, because the open nature of wireless networks makes them vulnerable to various attacks. It is also important for surveillance and Wireless Sensor Networks (WSNs). The jamming threat is particularly relevant in the context of mobile Remotely Piloted

Aircraft Systems (RPASs) and autonomous vehicles, as wireless communication is relied on for controlling the vehicle or receiving video feeds from the environment where the entities are located. When subject to jamming, such entities lose the communication link between them and their controllers. Before any action can be triggered as a countermeasure, the target entity has to detect jamming. Due to the simple and low-cost jamming strategy and its big effect on the wireless link there is an urgent need for jamming detection.
Detecting radio interference attacks is challenging as it involves discriminating between legitimate and adversarial causes of poor connectivity. Specifically, we need to differentiate a jamming scenario from other various network conditions: congestions that occur when the aggregated traffic load exceeds the network capacity so that the packet send ratio and delivery ratio are affected; the interruption of communication due to failures at the sender side; and other similar conditions.

We can generalize the jamming attack model using the following equations [3],

In the absence of jamming attacks, the received signal at location B is given by equation [1].

$$y(t) = x(t) + n(t) \qquad (1)$$

$x(t)$ is the desired signal.
$y(t)$ is the received signal.
$n(t)$ is an additive white Gaussian noise present within the path between the transmitter and the receiver.

While the receive signal at location B, in the presence of the jammer signal, is then given as in equation [2].

$$y(t) = x(t) + x_j(t) + n(t) + n_1(t) \qquad (2)$$

$x_j(t)$ is the jammer signal.
$n_1(t)$ is the noise within the path between the receiver and the jammer's location.

From this model we can get that if the relative jamming power is high enough it can cause high interference with the desired signal at the receiver causing the communication link to fail.

1.3. Previous Work in this Area

We can classify the previous work in detection of jamming as non-machine learning based and machine learning based. Traditional (non-machine learning based) techniques such as fuzzy logic, game theory, channel surfing, and time series analysis [2] were mostly proved to be inefficient in detecting smart jammers. Smart jammers typically operate by analyzing the incoming signals to identify the frequency, modulation, and other characteristics of the signal being transmitted by the enemy system. This can be done by generating a jamming signal that is synchronized with the enemy signal, or by using adaptive filtering techniques to selectively cancel out the enemy signal while leaving other signals intact.

The machine learning based techniques use metrics such as bad packet ratio, packet delivery ratio, received signal strength, ..., etc. [3]. Several machine learning algorithms were used as random forest, K-nearest neighbor, support vector machine and neural networks.
Although all these techniques achieve remarkably high detection accuracy and good performance, they can detect jamming only after the wireless communication link is completely corrupted and the remote control is lost with the entity i.e., the jamming significantly affected the communication link, thus at the jamming detection time, the target node cannot reliably communicate further, and it is required to autonomously take an action.

1.4. What is Jamming Early Detection and its Importance

Jamming early detection means to detect the presence of jamming before the communication link is compromised, thus, the base station is able to control the target node to take action and avoid the area in which jamming has an effect.
This solution can close the gap of the previous techniques and also can be considered as an anti-jamming technique. Early detection of jamming can be done by analyzing the raw I-Q samples of the signal acquired at the PHY layer and evaluating the shifting from the expected profile through Deep Learning (DL) techniques.

1.5. Problem Statement

As we discussed earlier, a jammer (intentionally) can completely disrupt the wireless communication link and accordingly entities lose the remote control. Our main target is to build a Deep Learning algorithm and train it using a transmitted dataset in the channel that has been subjected to jamming. The DL algorithm can early detect the Jamming presence that causes the poor connectivity and also classify its jamming type (Gaussian, Tone). This classification might help in removing its effect on the data.

Also optimizing this DL algorithm to be able to work on low power and battery running devices with minimum power consumption and maximum accuracy possible, because DL algorithms require a large computational power. Then we are required to test our model and our data on BladeRF X40 board.

1.6. Main Obstacle Found

The most challenging phase of building a deep learning model is the data set, a DL model needs a large dataset that has different scenarios to be considered as a practical model, we found a real dataset that is available online with jamming [4] but it has some issues and cannot be considered as a general dataset that can apply for other jamming attack scenarios. So, we decided to close this gap by generating our own dataset using the Python programming language to give the DL model the variety it needs to be able to early detect and classify the different jamming attacks. [4]

1.7. Chapters Overview

Chapter one gives a quick introduction about wireless networks, jamming definition and early detection and discusses the problem statement of the project as well as the previous work in the topic along with the main obstacle we had upon starting the project.

Chapter two illustrates the process of generating the dataset starting from the fundamentals of the communication system moving to the real dataset specifications that was available online.

Chapter three deals with the Deep Learning models we made starting by explaining what deep learning is and what its fundamentals are. It also discusses the CNN as well as LSTM with all their nuances, after that it gives a detailed illustrations of our models for real and generated dataset and quantization part ending with comparative analysis between different architectures of these models w.r.t performance and number of parameters.

Chapter four then continues with testing the models trained on the generated data using BladeRF board and evaluating the final models accordingly.

We then conclude our project and discuss the future work that can be done.

Chapter 2 Communication and Datasets

2.1.Communication System

Any communication system must have three components: transmitter, receiver and channel as shown in figure 1:

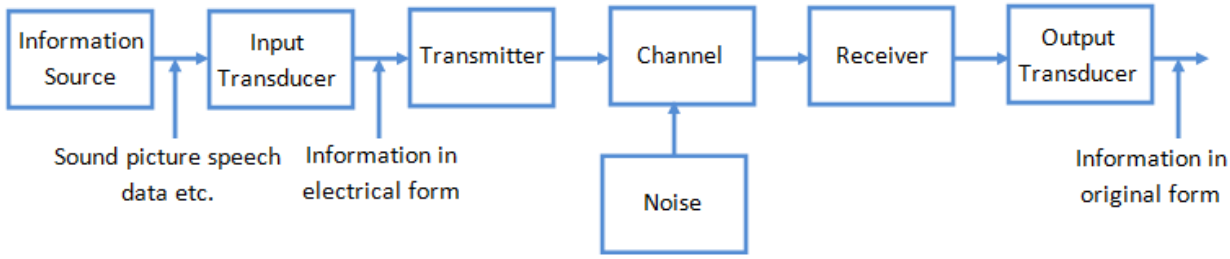

Figure 1: Communication System Components

Before transmitting the signals, they can undergo some operations as encoding, modulation, up sampling ... etc. and then the information is transmitted through the channel that can be wired or wireless. In the case of wireless channel, the data reaches the receiver through an unguided medium such as air. The channel can add some random noise from the environment, different types of fading … etc. when the data reaches the receiver it tries to cancel out these effects on the data by decoding, demodulating, down sampling … etc. i.e., inversion of the operations done at the transmitter to retrieve the original data.

1.7.1. Modulation and Demodulation

Modulation is the process by which data/information is converted into electrical/digital signals for transferring that signal over a medium. The process of extracting information/data from the received signal at the receiver side is called demodulation.

Many factors influence how accurately the extracted information represents the original input information. Electromagnetic interference can degrade the signal and make the original signal impossible to extract. Demodulators typically include multiple stages of amplification and filtering in order to eliminate interference.

1.7.2. Importance of Modulation

Modulation is used because it enables the ease of deployment, as the signal wavelength is inversely proportional to the frequency of the radiated signal and antenna size must be 1/10th of the wavelength. For example, if the used frequency is 5KHz, the wavelength is 60,000 meters, so the antenna size would be around 6 kms. In that case it is quite impossible to set up an antenna of that size. So, by using the modulation technique the size of the antenna is reduced by transmitting the signal on a high frequency.
Also, the modulation enables sending multiple signals simultaneously or multiplexing by FDM, TDM, ... etc.

1.7.3. How Modulation Works

Modulation is done by imposing an input signal onto a carrier, in other words modulation changes the shape of a carrier wave to somehow encode the speech or data information that we were interested in carrying. Modulation is like hiding a code inside the carrier wave. This process is done mainly by multiplying the signal by a carrier with high frequency

1.7.4. Types of Modulation

As shown in figure 2, there are many types of modulation schemes that are used in communication systems, the following chart represents the different types of modulation, in our project we will focus on Binary phase shift keying (BPSK) in digital modulation PSK branch.

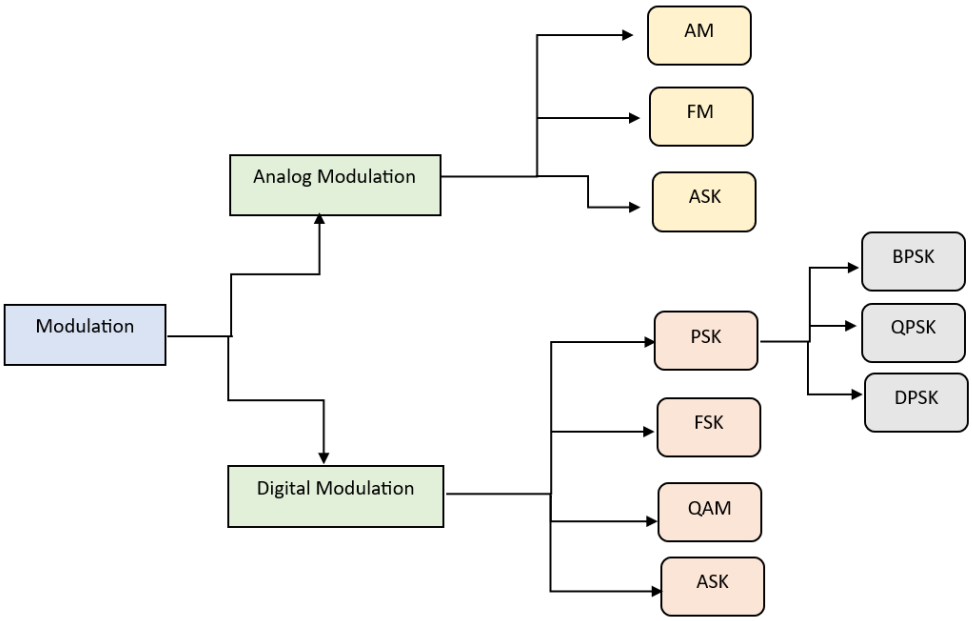

Figure 2: Types of Modulation

1.7.5. Digital Modulation

Digital modulation is the process of encoding a digital information signal into the amplitude, phase, or frequency of the transmitted signal. The encoding process affects the bandwidth of the transmitted signal and its robustness to channel impairments. In general, a modulation technique encodes several bits into one symbol, and the rate of symbol transmission determines the bandwidth of the transmitted signal. Since the signal bandwidth is determined by the symbol rate, having a large number of bits per symbol generally yields a higher data rate for a given signal bandwidth. However, the larger the number of bits per symbol, the greater the required received SNR (signal to noise ratio) for a given target BER (bit error rate).

1.7.6. Why BPSK

Binary-phase-shift-keying (BPSK) is a widely used modulation scheme in wireless communications due to its simplicity and robustness. It is a digital modulation scheme that transmits information by varying the phase of a carrier signal between two discrete states, typically represented by 0 and 1. BPSK is particularly attractive for jamming detection purposes because it provides a clear distinction between the two states, making it easier to identify deviations caused by jamming techniques. By focusing on a simple modulation scheme like BPSK, we can reduce the complexity of the detection task and potentially achieve higher accuracy in classifying jamming signals.

Additionally, BPSK is less susceptible to interference and noise compared to more complex modulation schemes. In wireless communication systems, noise and interference are common challenges that can degrade the signal quality and affect the performance of jamming detection algorithms. By using BPSK, we can reduce the impact of noise and interference since the detection primarily relies on the phase information, which is relatively more resistant to such disturbances. This robustness is particularly beneficial when dealing with real-world scenarios where the received signal may be subject to various environmental conditions and interference sources.

Most importantly, if the proposed detection techniques work on a BPSK system, it might work on higher modulation schemes. If not, then it probably won’t work on them as well.

1.7.7. I-Q Samples

In wireless communication, I-Q representation, also known as In-phase and Quadrature-phase representation, is a method of representing a complex signal as two separate signals: one in-phase (I) with the carrier signal and another quadrature-phase (Q) with the carrier signal.

By analyzing the I and Q components, we can get a better understanding of the signal's properties and characteristics.

1.7.8. Using I-Q Samples for Jamming Detection

I-Q samples are commonly used in anomaly detection methods as they help extract information about the signal fairly quickly. They also capture structures that are not present in other signal information like SNR and power spectrum.

For example, in a jamming scenario, if an attacker try to mask their interference by mimicking the power spectrum of the legitimate signal, when we try to analyze the power spectrum of the received signal then we will not recognize the jamming as we will see the expected received signal power spectrum, so in this case, even if the jamming with hidden in the power spectrum, if we analyze the IQ samples, it would be possible to detect changes in phase and amplitude of the signal which would not be seen in the power spectrum analysis.

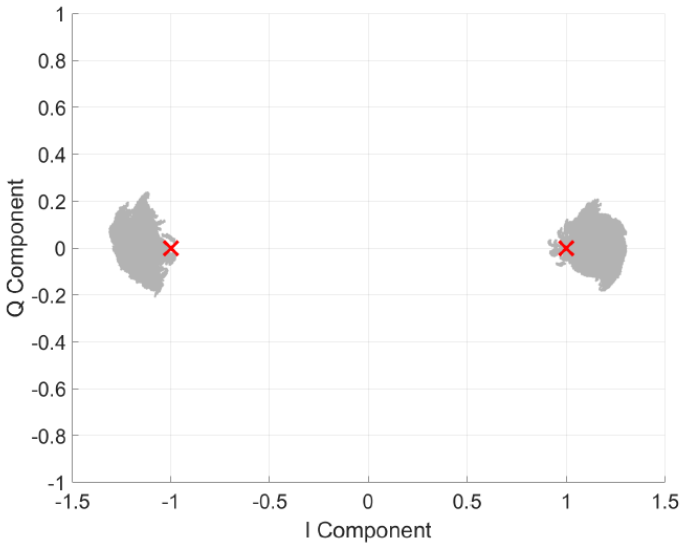

Figure 3: BPSK Symbols [4]

2.2. Dataset Specifications

The following part describes the specifications of two datasets that we worked on. The first one is a real-life dataset provided in [4] that was captured using the Software-defined Radio setup of the USRP Ettus Research x310. The second dataset is a dataset that we generated using the Python programming language.

2.2.1. USRP Real Dataset Specifications

2.2.1.1. Experiment Statement

In this paper, a collection of actual indoor communication scenarios influenced by various jamming methods is introduced. The dataset comprises data obtained from seven distinct Software Defined Radios (SDRs), specifically the USRP Ettus Research X310, operating within an office setting. Each experiment consists of a transmitter, a receiver, and a jammer [4].

2.2.1.2. Hardware Setup

The setup consisted of three radios (receiver, transmitter, and jammer) connected to USRP Ettus Research x310 hardware via Ethernet. The VERT2450 omnidirectional stylo antenna was used in an office environment. The transmitter and the jammer were placed close together, while the receiver's location varied. The article explains the measurement setup used for data collection and analysis of jamming effects on wireless communication systems. It involved seven Ettus Research x310 SDRs with ubx160 daughterboards and a vert2450 omni-directional stylo antenna. Devices 2 and 3 served as the transmitter and receiver, respectively, while devices 4 to 8 acted as jammers. The SDRs were connected to Dell xps15 9560 laptops running the GNU Radio Development Toolkit. The experiments were conducted in an indoor office environment during working hours, experiencing dynamic channel conditions with non-line-of-sight and multipath effects (which complicates the detection process).

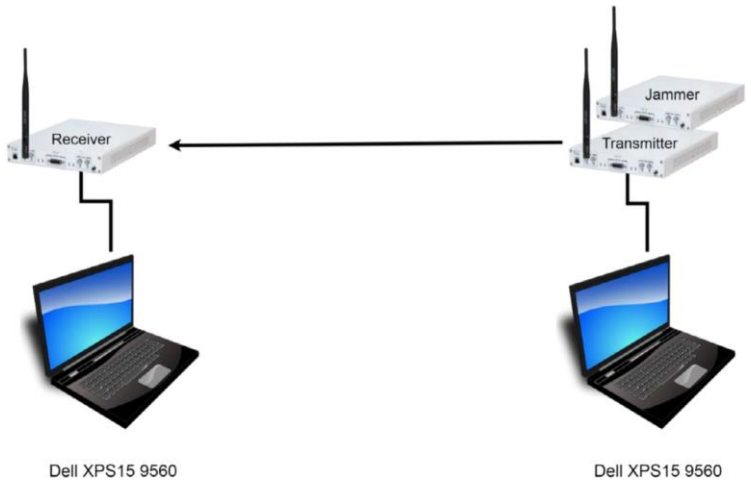

Figure 4: Hardware Setup For Data Collection [4]

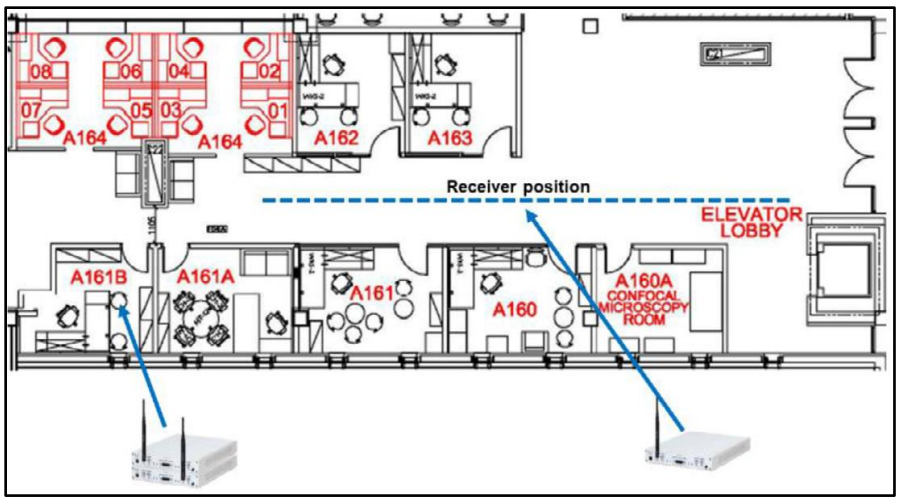

Figure 5: The Office Environment Which Hold The Experiments [4]

2.2.1.3. Software Setup

The article discusses the software platform used in their experiments, which is based on the GNURADIO development toolkit. The experiments were conducted using a reference frequency of 900 MHz for both communication and jamming, with a sample rate of 1 million samples per second. The transmission power and receiver gain were normalized to 1, which corresponds to 15dbm (32mw) for the transmission power. The receiver chain includes an adaptive gain control (AGC) block, a symbol sync block, a Costas loop block, and a constellation decoder block. The jammer includes a signal source and a USRP sink block. The experiments involved placing the transmitter (+ the jammer) and the receiver at different distances and changing the relative values of the jammer's transmission power between 0 and 0.8 to investigate the effect of the jamming. It also involved changing the jamming device used.

The transmitter used the constellation modulation block to encode and send a sequence of 256 bytes using binary phase shift keying (BPSK) modulation. The receiver used an adaptive gain control block, a symbol sync block, and a Costas loop block to recover the center frequency of the carrier and down-convert the signal to baseband. The jammer used a noise signal source to generate two different waveforms or a digital gaussian distributed sequence, and a USRP sink block to transmit the jamming signal to the USRP device. The GNU radio block diagram for the jammer included a signal source and a USRP sink.

2.2.1.4. Data Description

The dataset consists of 31 .mat files, each containing three matrices representing the different jamming modes: "No-jamming," "Sine," and "Gaussian." each matrix contains two columns representing the I and Q samples, respectively. Each matrix contains around 150 million samples.

The experiments involve varying:

1. The intensity of the jamming signal:

Measurements from w1 to w18 refer to fixed distance between Tx and Rx (10 m), 2 different jammers (id 4 and 5), and varying the relative jamming power between 0.1 and 0.8.

2. The jamming device:

Measurements from w19 to w23 refer to relative jamming power (0.5), distance (10 m), Tx (id = 2) and Rx (id = 3) are fixed, jamming devices (id spanning from 4 to 8).

3. Distance between the transmitter and receiver:

Measurements from w24 to w31 refer to fixed jamming power to 0.5, Tx, Rx and jamming ids are 2, 3, and 4, respectively, distance changes between Tx and Rx in 3, 5, 7, 10, 13, 16, 19 and 21 meters.

Table 1 shows the experiment data files.

Table 1: Files Included in The Dataset Along with The Specific Experiment Parameters.

File Name	*Jammer Id*	*Relative Jamming Power*	*Distance Between Tx and Rx [Meters]*
W1	4	0.1	10
W2	4	0.3	10
W3	4	0.6	10
W4	5	0.1	10
W5	5	0.3	10
W6	5	0.6	10
W7	4	0.2	10
W8	5	0.2	10
W9	4	0.4	10
W10	5	0.4	10
W11	5	0.5	10
W12	4	0.5	10
W13	4	0.7	10
W14	5	0.7	10
W15	4	0.8	10
W16	5	0.8	10
W17	5	0.6	10
W18	4	0.6	10
W19	4	0.5	10
W20	5	0.5	10
W21	6	0.5	10
W22	7	0.5	10
W23	8	0.5	10
W24	4	0.5	3
W25	4	0.5	5
W26	4	0.5	7
W27	4	0.5	10
W28	4	0.5	13
W29	4	0.5	16
W30	4	0.5	19
W31	4	0.5	21

Figures [6-8] represent some of the USRP dataset constellation diagrams for the 3 classes (No-jamming, Gaussian Jamming, and Sine jamming respectively), for one frame and the entirety of the dataset at a relative jamming power of 0.8

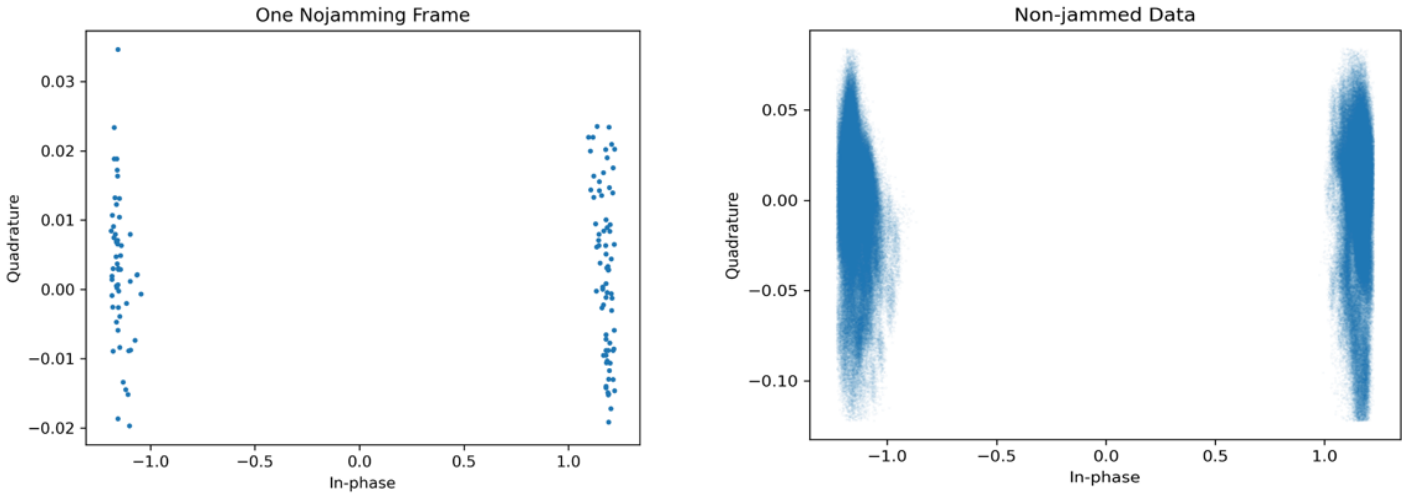

Figure 6: USRP No jamming Constellation Diagram

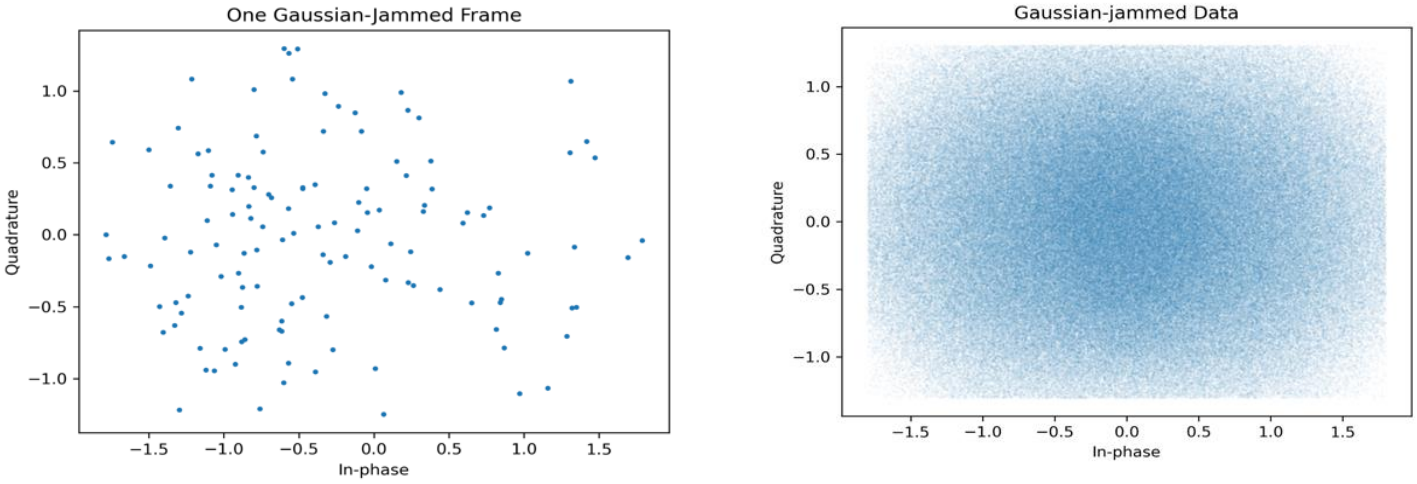

Figure 7: USRP Gaussian Jamming Constellation Diagram

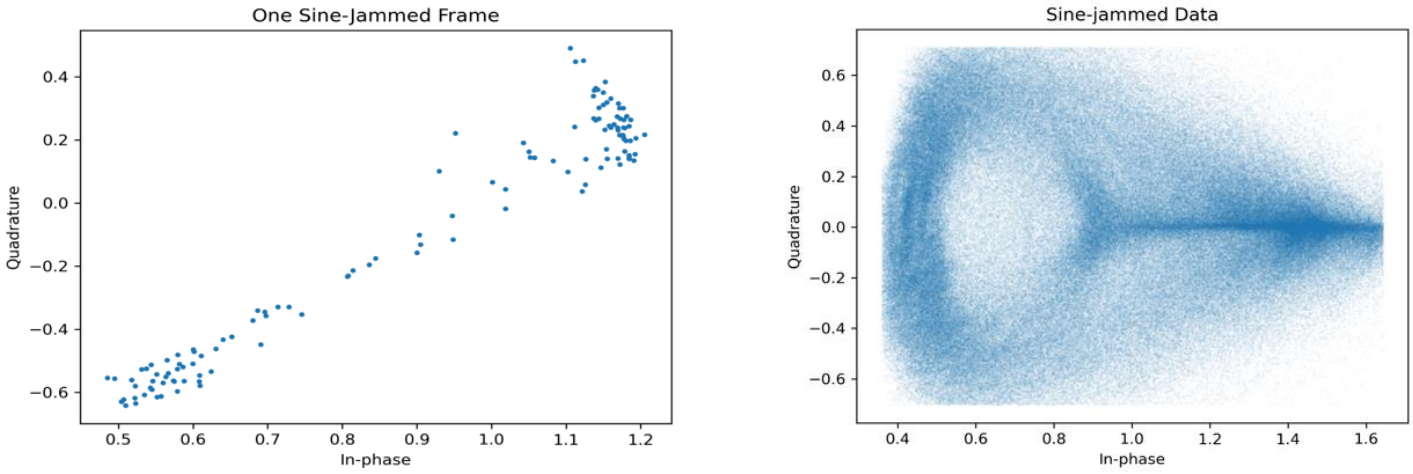

Figure 8: USRP Sine Jamming Constellation Diagram

2.2.2. Generated Dataset Specifications

The following part describes the generated dataset and its specifications. This dataset was generated due to the fact that the available dataset provided in [4] displays characteristics of high SNR transmissions. As will be discussed later, training the models on it results in them not being able to differentiate possible poor channel conditions (like high noise) from a jamming attack. So, generating this dataset to train the models on it will solve this issue.

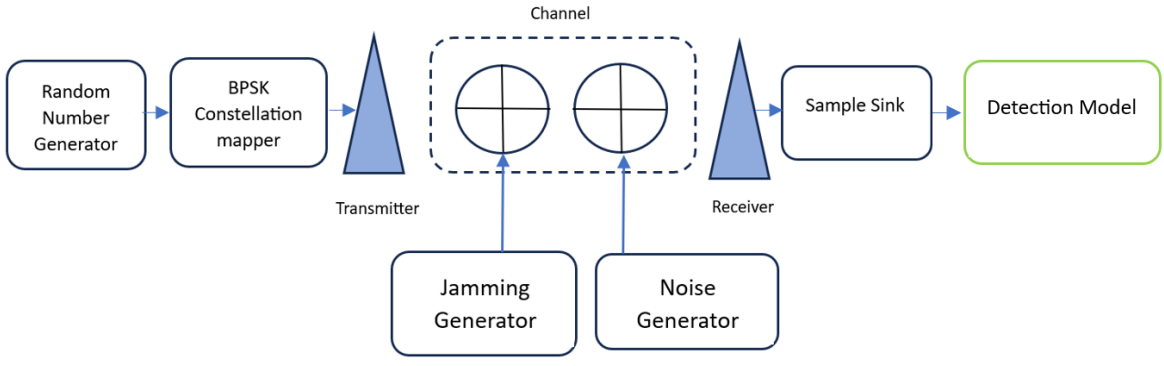

Figure 9: Communication system block diagram with jamming generation

The block diagram in figure [9] describes the communication system that was simulated for the generation of the dataset. It consists of the following blocks:

1. Random number generator: generates the data bit stream to be transmitted.
2. BPSK Constellation mapper: maps the bit stream into BPSK symbols.
3. Channel: where the BPSK signal, jamming signal, and noise are added together.
4. Jamming Generator: represents the jamming device.
5. Noise Generator: AWGN generator.
6. Sample Sink: where the received samples are collected
7. Jamming early detection model: where the detection model operates.

2.2.2.1. Communication System Specifications

The communication system was entirely simulated using the Python programming language and its various libraries. At the transmitter, random BPSK symbols are generated and transmitted with 0 dB transmitting power, and then the noise is added to account for the desires SNRs at the receiver in the absence of jamming. The channel noise is modelled as additive white gaussian noise (AWGN). Equation (3) describes how the noise is added to the transmitted signal:

$$Noisy\ Signal = Signal + \sqrt{\frac{E_s}{2\ SNR_{linear}}} N(0,1) \qquad (3)$$

Where $N(0,1)$ is a normal distribution of mean=0 and variance=1. [5]

This process is used to generate a certain number of I/Q samples that are then divided into frames; each frame consists of 128 I/Q samples.

2.2.2.2. The Jamming Scenarios Generated

There are three scenarios that are explored:

1. No-jamming
   BPSK samples at which there is no active jamming to the signal, and the only change done to the I/Q samples is the AWGN.
2. Gaussian-jamming
   A type of jamming that consists of samples that are pulled from a standard gaussian distribution.
3. Tone-jamming
   A type of jamming that consists of a single tone jamming at a certain frequency.

These three scenarios are what we will focus on in our project, the following sections will illustrate these three scenarios in detail.

1. Non-jammed Data

50,000 frames in total are generated, and AWGN is added to them with a different SNR for each 10,000. The SNR values used are **10, 20, 30, 40, and 50 dB**. This range of SNR values is used so that the DL model learns to detect a proper BPSK signal even at non-ideal channel conditions, as a link with SNR of 10 dB and less is typically considered a poor-quality link, while 40 dB and above is considered a high-quality link. [15]

Figures [10-12] show the constellation diagrams of one frame of the non-jammed data and the constellation diagrams of the entirety of the non-jammed dataset at different SNR values.

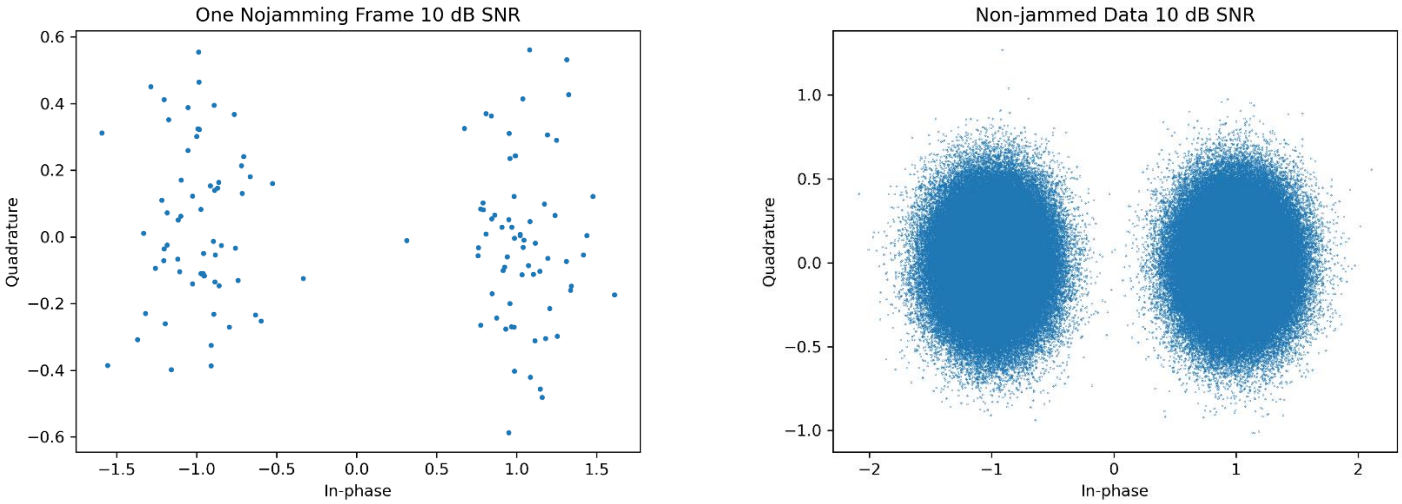

Figure 10: Non-jammed Data, 10 dB SNR

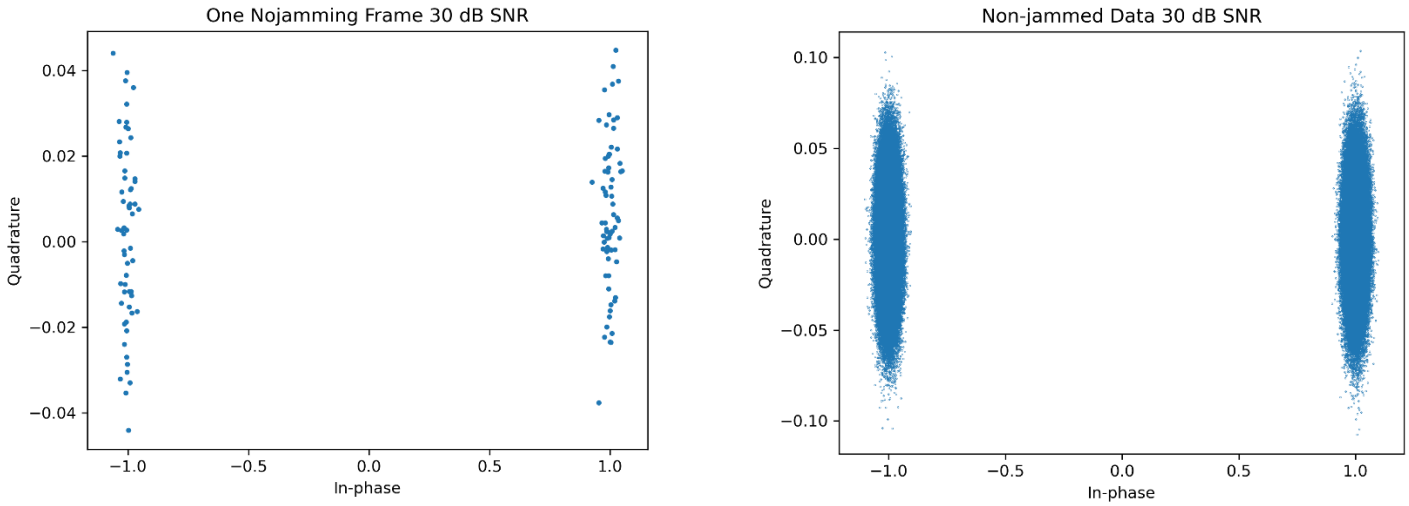

Figure 11: Non-jammed Data, 30 dB SNR

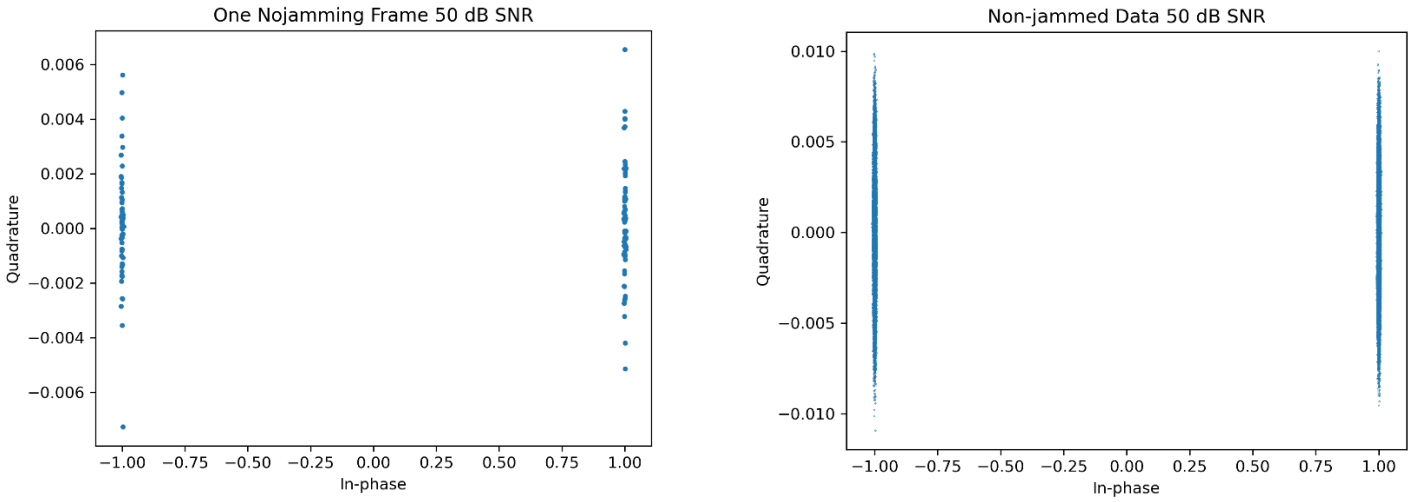

Figure 12: Non-jammed Data, 50 dB SNR

2. Gaussian-jammed Data

In this scenario, a jamming device transmits samples that are pulled from a standard gaussian distribution (similar to AWGN) with different values for the transmitting power of the jamming device.

50,000 frames in total are generated, 1000 frames for each combination of SNR and Jamming Power values. The SNR values used are **(10, 20, 30, 40, 50) dB**, which are the same as the No-jamming case, and the jamming power values used are **(-4, -3, -2, -1, 0, 1, 2, 3, 4, 5) dB**. Thus, 50 scenarios are generated for the gaussian jamming type. The range of jamming powers is used so that the model learns to detect the jamming **early;** when the effect of jamming can be considered negligible (0 dB and below), as well as when it's detrimental to the communication link (above 0 dB). Equation (4) describes how the jamming signal is added to the noisy transmitted signal.

$$Jammed\ Signal = Noisy\ Signal + \sqrt{\frac{P_j}{2}}N(0,1) \qquad (4)$$

Where P_j is the jamming power (linear). [6]

Figures [13 -15] show the constellation diagrams of one frame of the Gaussian-jammed data and the constellation diagram of the entirety of the Gaussian-jammed dataset at SNR= 50 dB at different jamming power values (to show the effect of jamming on the constellation).

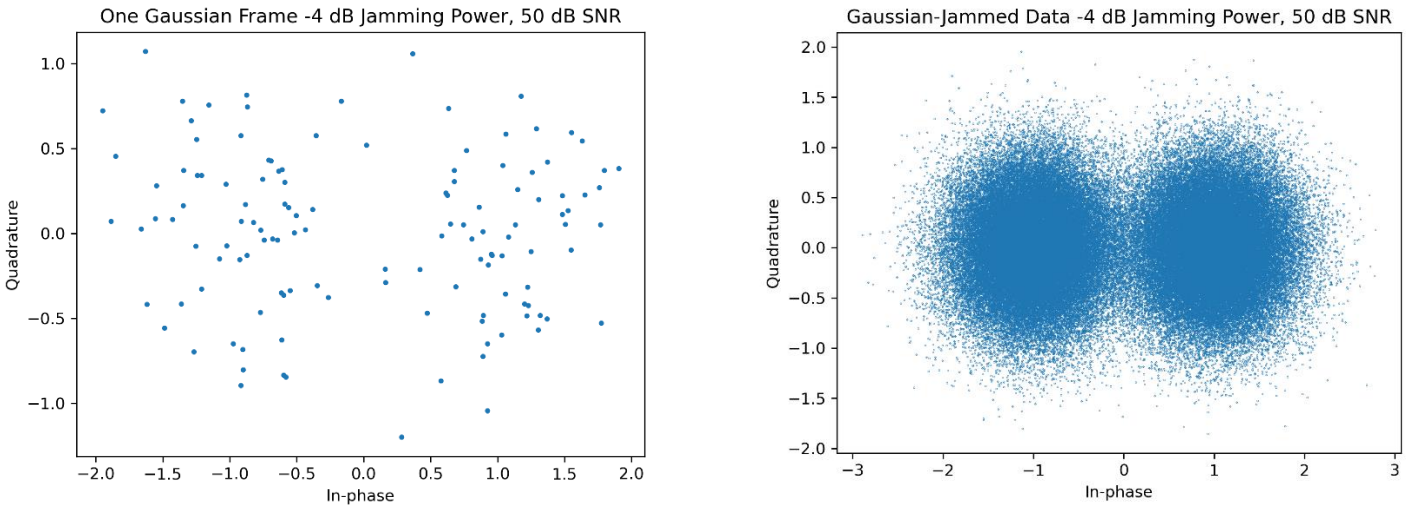

Figure 13: Gaussian-jammed Data, Pj = -4 dB

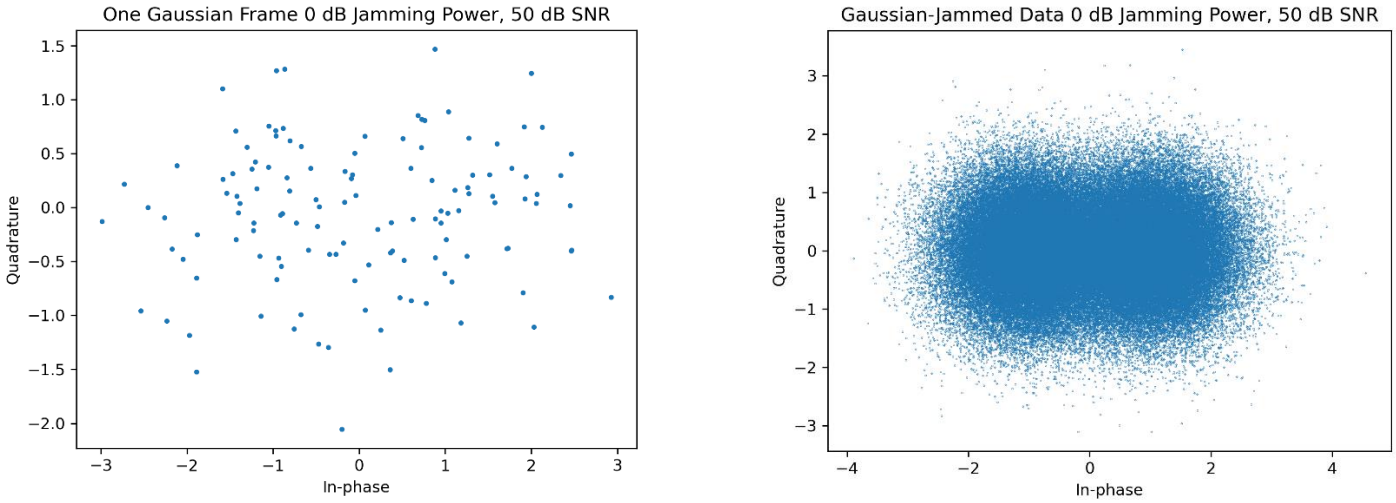

Figure 14:Gaussian-jammed Data, Pj =0 dB

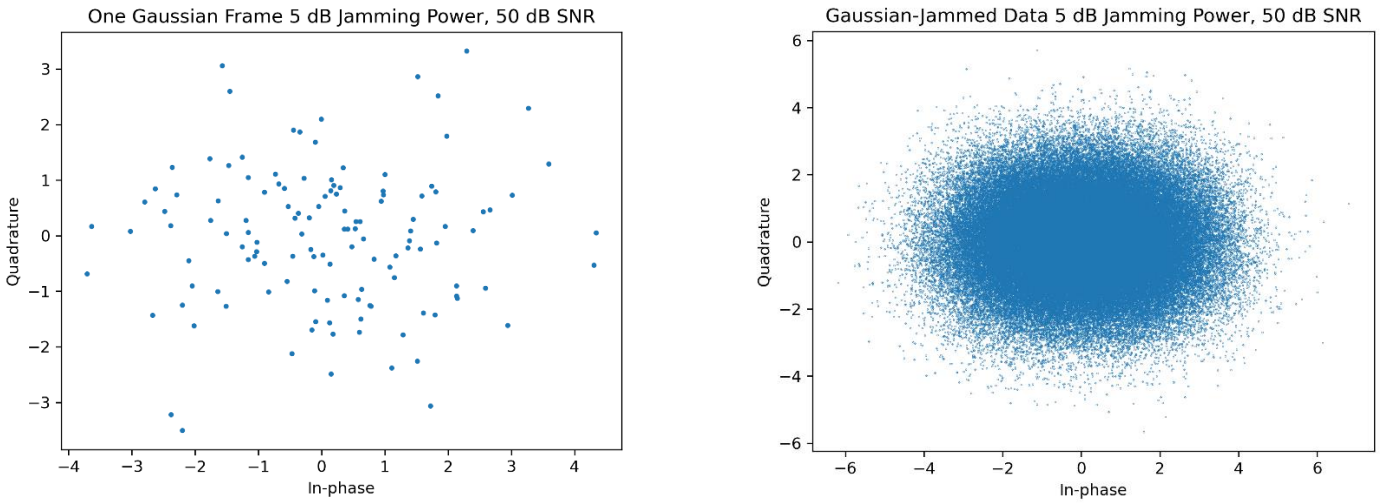

Figure 15: Gaussian-jammed Data, Pj = 5 dB

3. Tone-jammed Data

In this scenario, a jamming device continuously transmits a single tone that changes frequency and phase every certain period. In this case, that period is equivalent to the time it takes 128 symbols (one frame) to be transmitted from the transmitter to the receiver. Like the Gaussian-jamming scenario, 50,000 frames in total are generated, 1000 frames for each combination of SNR and Jamming Power values. Thus, °0 tone jamming possible scenarios are generated.

Equation (5) describes the jamming tone.

$$Tone = \sqrt{P_j}\, Cos\left(2\pi\, f_j\, t + \varphi_j\right) \quad (5)$$

Where P_j is the jamming power (linear).

f_j is the frequency

φ_j is the phase of the tone signal. [6]

Figures [figure numbers 16-18] show the constellation diagrams of one frame of the Tone-jammed data and the constellation diagram of the entirety of the Tone-jammed dataset at SNR= 50 dB at different jamming power values. For context, a jamming power of 5 dB results in around **30% BER** (in both jamming scenarios) as seen in figure 19.

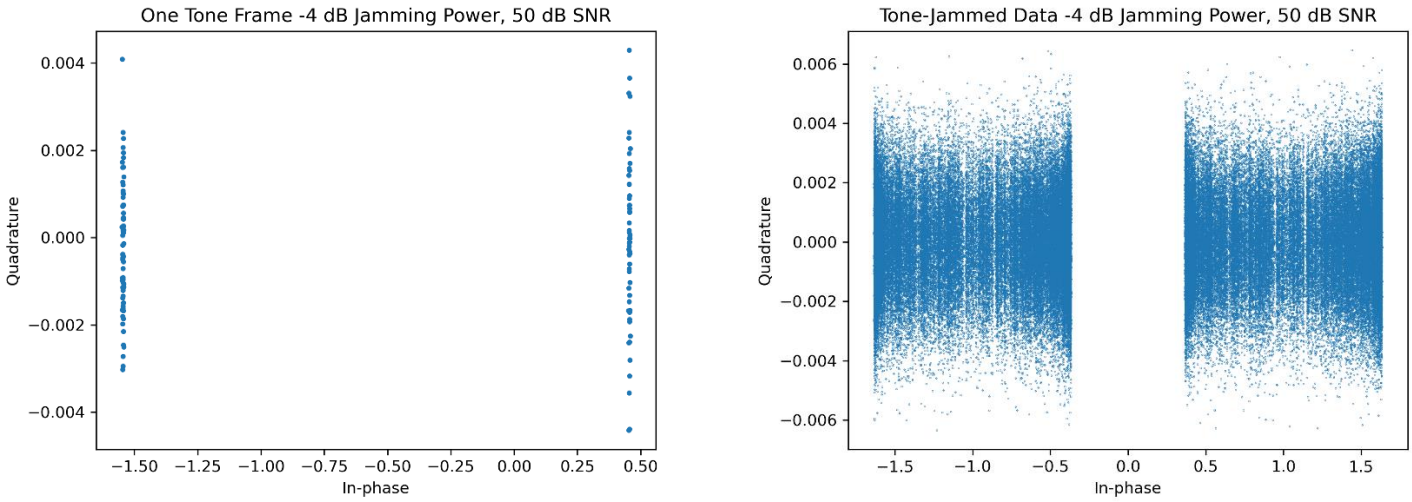

Figure 16: Tone-jammed Data, Pj = -4 dB

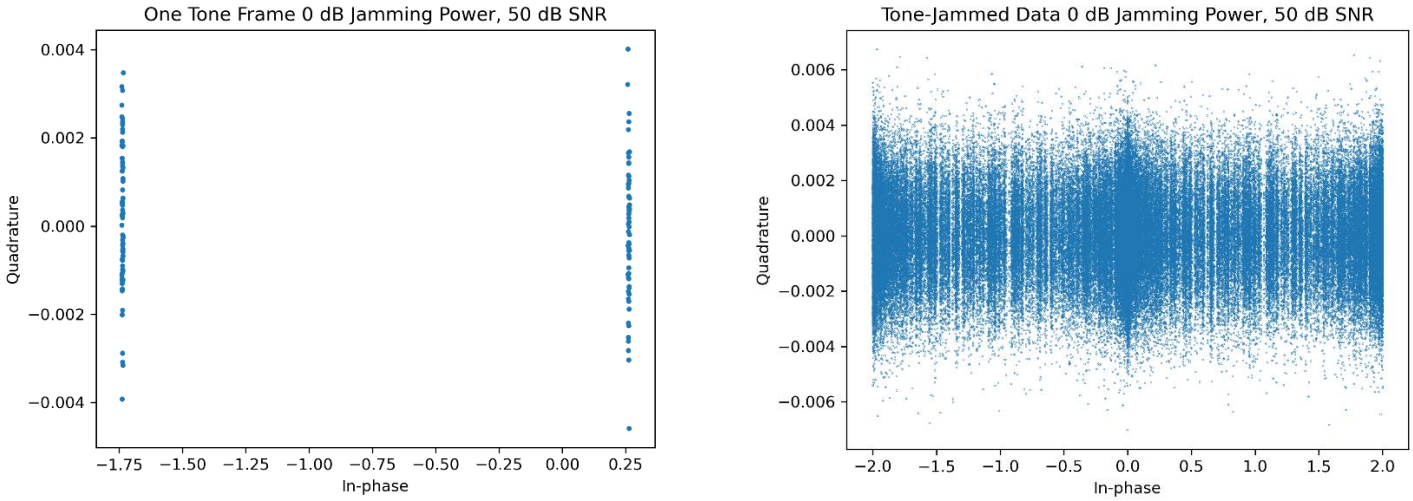

Figure 17: Tone-jammed Data, Pj= 0 dB

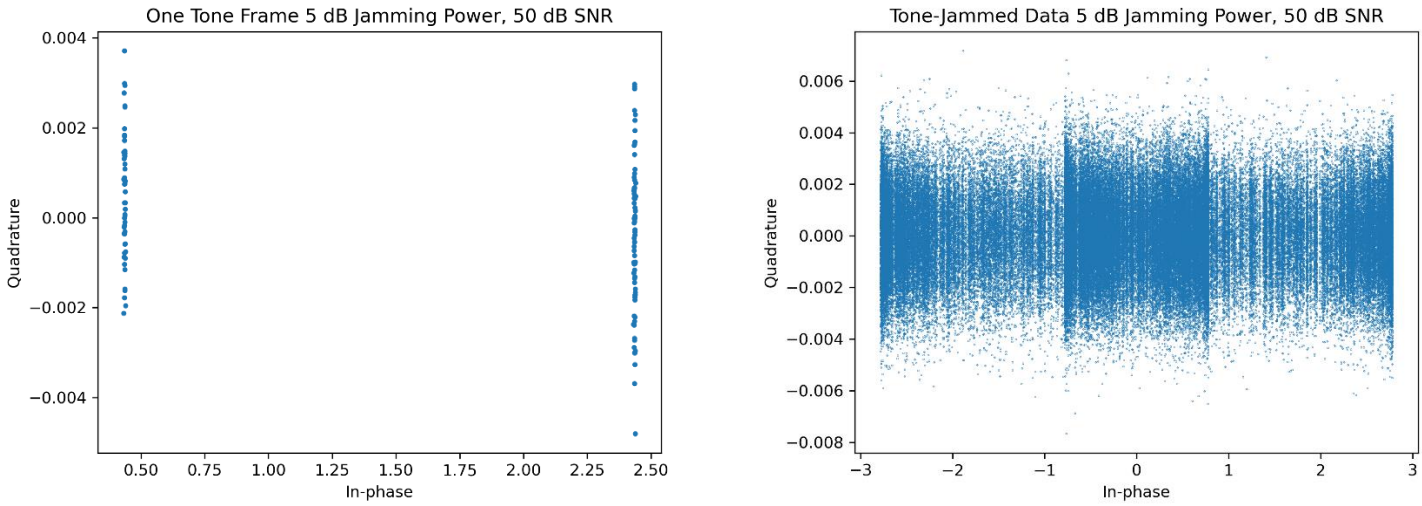

Figure 18: Tone-jammed Data, Pj =5 dB

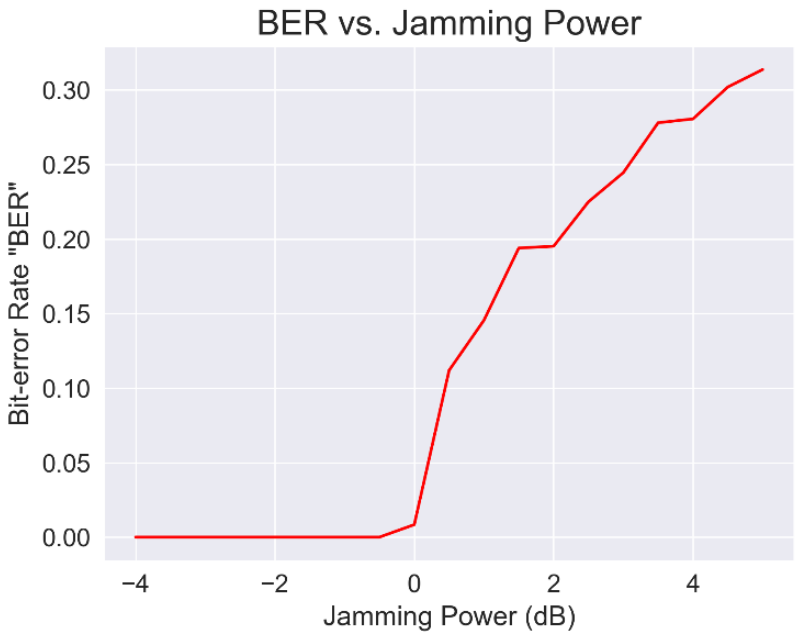

Figure 19: Generated Dataset BER vs. Jamming Power

Chapter 3: Deep Learning

3.1.Fundamental Concepts

3.1.1. Neural Networks Overview

Deep learning is a subfield of machine learning which enables the machine to perform human-like tasks without human involvement as it is concerned with algorithms inspired by the structure of the brain (the so-called neural networks), hence the name artificial neural networks (figure 20).
Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are at the heart of deep learning algorithms. Artificial neural networks (ANNs) are comprised of multiple layers of nodes called "neurons". These layers are usually an input layer, one or more hidden layers, and an output layer. Each neuron connects to another and has an associated weight and threshold. If the output of any individual neuron is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.
Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high speeds. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts.

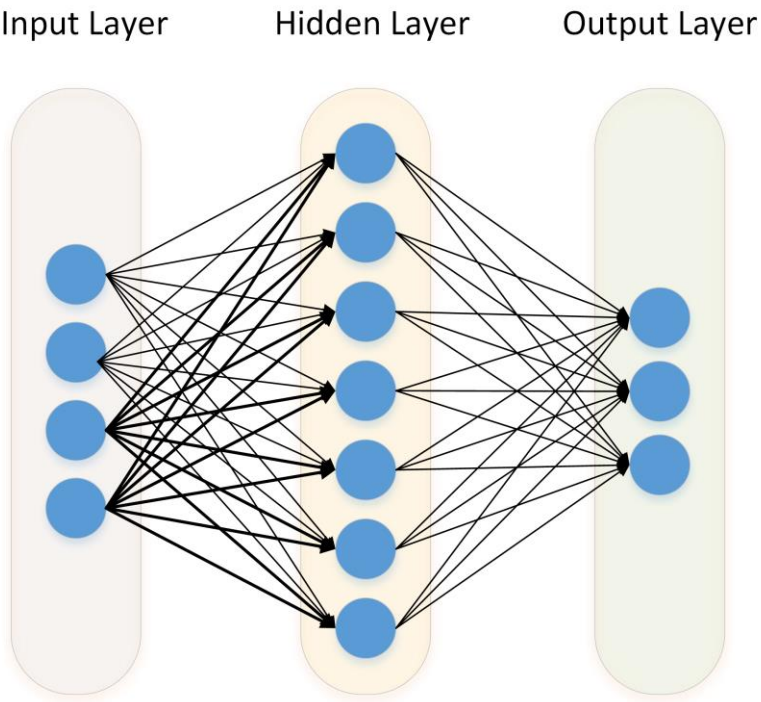

Figure 20: A Neural Network

3.1.2. What are Convolutional Neural Networks (CNNs)

A CNN is a type of neural network commonly used in computer vision tasks such as image classification, object detection, and segmentation. What sets CNNs apart from other neural networks is their ability to automatically learn spatial hierarchies of features from images by using convolutional layers. These layers apply a set of filters (kernels) to the input image, which extracts features such as edges, shapes, and textures. CNNs also use pooling layers to downsample the feature maps, reducing the spatial dimensions of the input and making the network more computationally efficient. Finally, fully connected layers are used to classify the input into different categories. One of the advantages of CNNs over traditional machine learning algorithms is their ability to automatically learn features from raw image data, which can lead to better classification accuracy. Additionally, CNNs are able to handle large datasets more efficiently than traditional algorithms, making them well-suited for tasks such as image recognition on large datasets.

3.1.2.1. Convolutional Layer in CNN

The most important component of CNN architecture is the convolutional layer, which uses a collection of convolutional filters or kernels to generate an output feature map by convolving with the input image. Each kernel is defined as a grid of discrete numbers, or kernel weights, which are randomly assigned at the beginning of training and adjusted at each training era to extract significant features. The convolutional operation involves sliding the kernel over the input image, calculating the dot product between the kernel and corresponding values of the image to generate the feature map. Padding and stride values can be used to adjust the size of the output feature map. The core benefits of convolutional layers include sparse connectivity and weight sharing, which reduce the number of required weights and connections, and decrease training time and costs. Figure 21 shows how the kernel is applied to the input matrix.

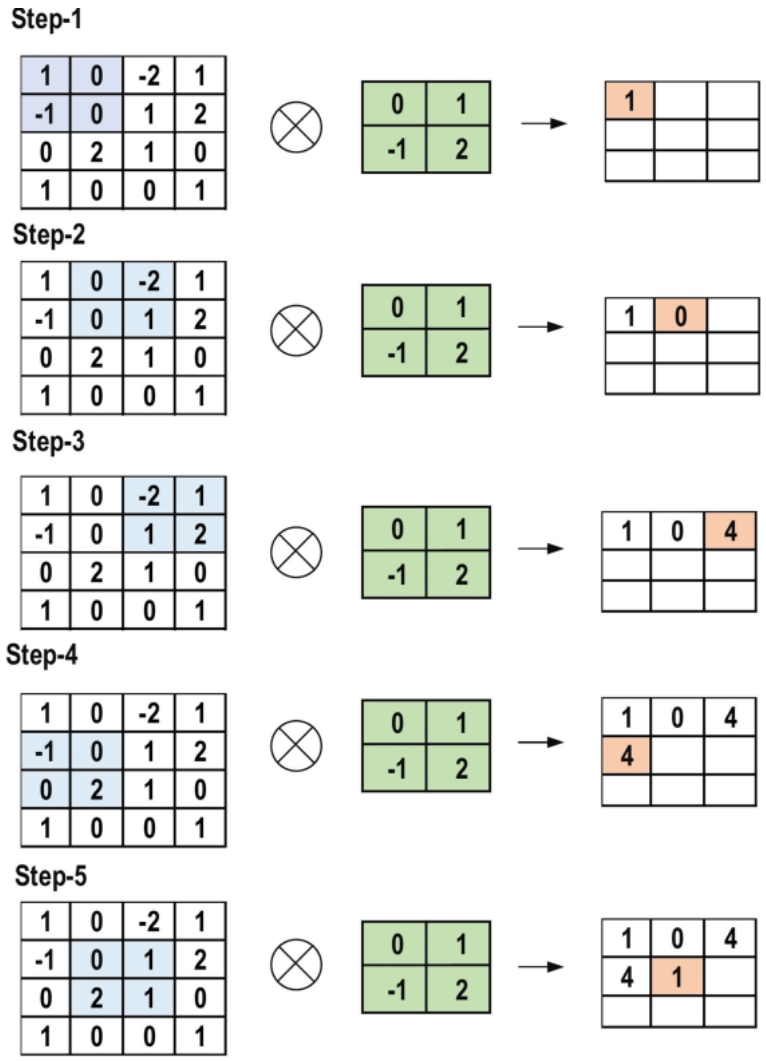

Figure 21: Example of the primary calculations executed at each step of convolutional layer

3.1.2.2. Padding

Padding (figure 22) adds zeros around the edges of an input tensor to increase its spatial dimensions. It is used in convolutional layers to avoid losing information at the edges of the input and its purpose is to preserve the spatial dimensions of the input tensor. A "same" padding is a type of padding used in convolutional layers and it results in the output having the "same" dimensions as the input of the layer.

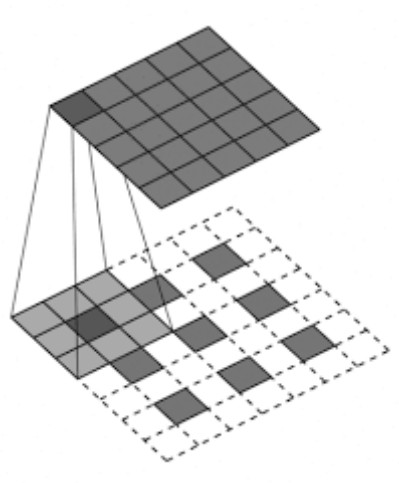

Figure 22: Padding

3.1.2.3. Stride

The stride (figure 23) is a parameter that determines the step size of the filter when it is applied to the input tensor. The stride controls the size of the output tensor and the receptive field of the convolutional layer. A larger stride reduces the size of the output tensor and increases the receptive field, while a smaller stride produces a larger output tensor and a smaller receptive field. The stride parameter is commonly used in convolutional layers to reduce network computational cost and speed up training and inference.

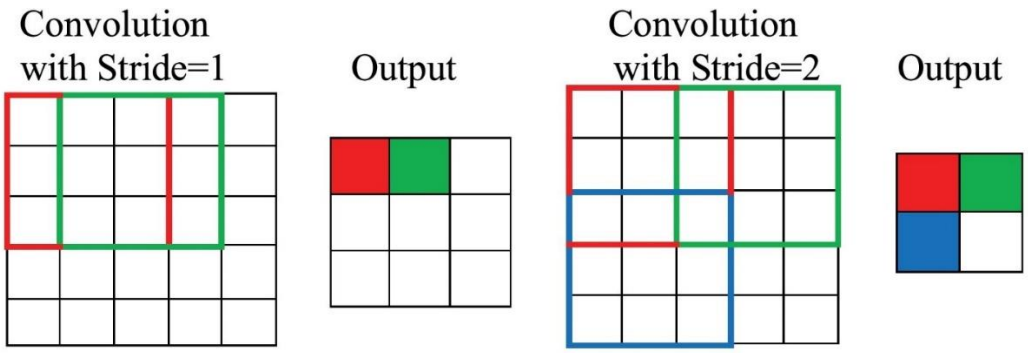

Figure 23: stride

3.1.2.4. Pooling Layer

The pooling layer is responsible for down sampling the feature maps created through convolutional operations by reducing their size while retaining the most critical information. Stride and kernel sizes are assigned before executing the pooling operation, and there are various types of pooling methods available, such as max, min, and global average pooling shown in figure 24. However, the pooling layer can sometimes negatively affect the overall performance of the CNN by solely

focusing on identifying the correct location of a feature, potentially causing the model to miss relevant information in the input image.

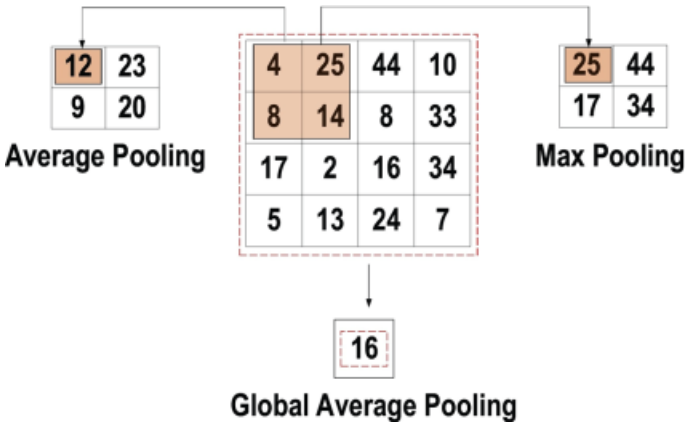

Figure 24: Example of maximum, average and global average pooling.

3.1.3. Activation Functions

Activation functions are crucial in neural networks as they map input to output and determine whether a neuron should fire based on the input. Non-linear activation layers are used in neural networks after learnable layers, like convolutional and fully connected layers, to enable the network to learn complex features. The activation function must also be differentiable to allow for error back-propagation during training. Popular activation functions include sigmoid, tanh, ReLU and SoftMax.

3.1.3.1. ReLU

ReLU (figure 25) stands for Rectified Linear Unit, which is the most commonly used activation function in neural network hidden layers. It is a simple and computationally efficient function that operates element-wise on the input. The ReLU function takes an input u and returns the maximum of 0 and u. In other words, if u is negative, the output is 0, and if u is positive or zero, the output is u. Mathematically, the ReLU function can be represented as follows:

$$f(u) = max(0, u) \qquad (6)$$

The ReLU function has several advantages that make it popular in neural network models. Firstly, it is computationally efficient and easy to implement. Secondly, it has a sparsity property, which means that it can set some of the input neurons to zero, making the network more efficient and less prone to overfitting. Finally, ReLU has been shown to perform well in practice, especially in deep neural networks.

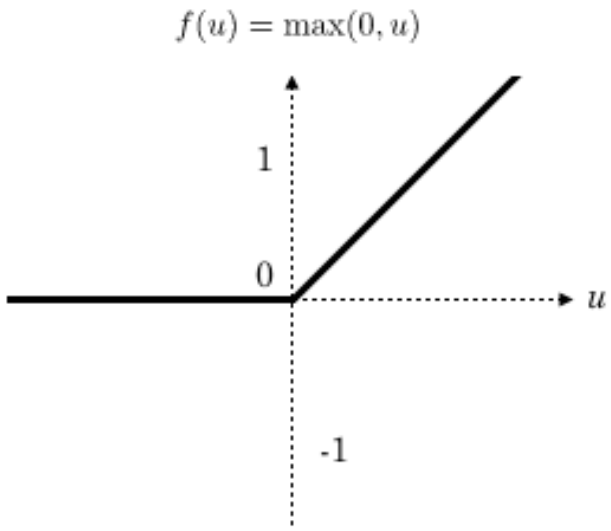

Figure 25: ReLU function

3.1.3.2. SoftMax

SoftMax is the most used activation function in the output layer of neural networks that work on multi-class classification problems. It generates an output probability p ∈ {0, 1}. It is used in the output layer to produce a probability-like output. The mathematical representation of the output class probability can be seen in Equation [7]. During training, the SoftMax function is typically used in conjunction with a cross-entropy loss function to measure the difference between the predicted and actual output for a given input, and the goal is to minimize the loss function by adjusting the weights and biases of the neural network.

$$p_i = \frac{e^{a_i}}{\sum_{K=1}^{N} e^{a_k}} \quad (7)$$

3.1.3.3. Sigmoid

The sigmoid activation function (figure 26) is a mathematical function commonly used in artificial neural networks. It takes an input value and maps it to a value between 0 and 1. The sigmoid function has an S-shaped curve, starting at 0 and gradually increasing towards 1 as the input becomes larger. This characteristic makes it particularly useful for binary classification tasks, where the output can be interpreted as a probability. The sigmoid function is differentiable and monotonically increasing, allowing for gradient-based optimization algorithms to be applied during the training of neural networks. Equation 8 the function.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (8)$$

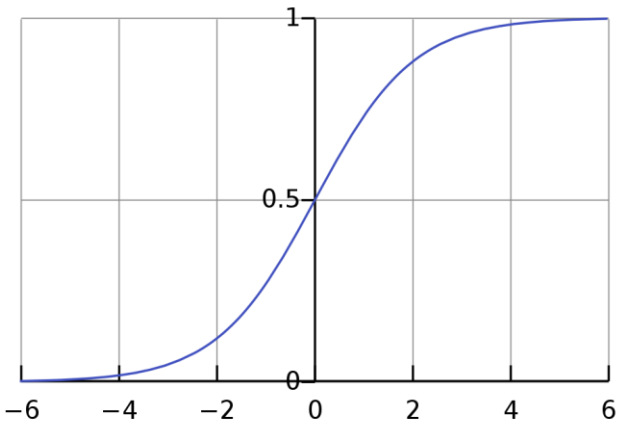

Figure 26: Sigmoid Activation Function

3.1.4. Flatten layer

A flatten layer converts a multidimensional input tensor into a one-dimensional vector as shown in figure 27. The flatten layer is typically used between the convolutional layers and the fully connected layers in a CNN. When a convolutional layer produces a feature map as an output, the flatten layer reshapes the feature map into a one-dimensional vector. This vector can then be used as input to a fully connected layer, which computes a weighted sum of the inputs and applies an activation function to produce an output value. The flatten layer does not contain any trainable parameters. Its purpose is to reshape the input tensor into a format that can be processed by the fully connected layer.

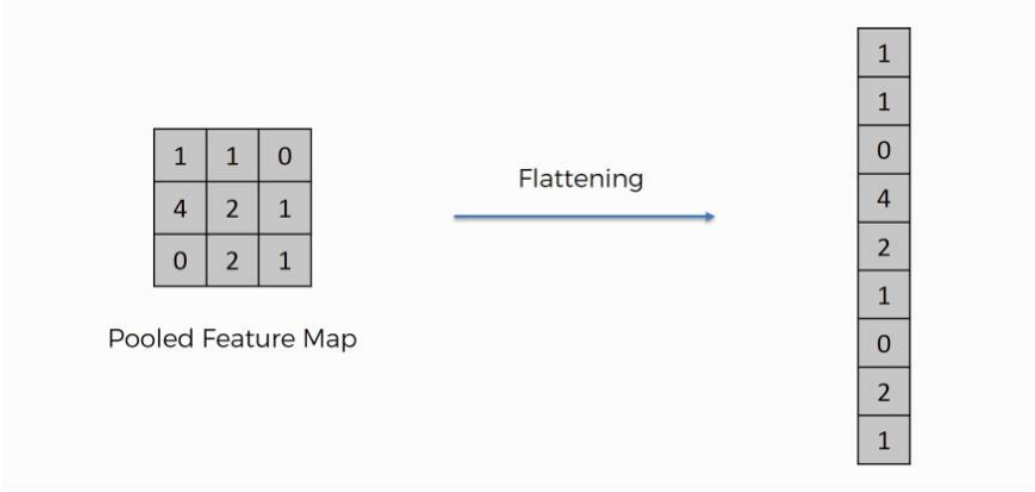

Figure 27: Flattening Example

3.1.5. Fully Connected Layer

The Fully Connected Layer (also known as the **Dense** Layer) is typically located at the end of a CNN architecture and also serves as the model classifier. In this layer, each neuron is connected to all neurons in the previous layer, using a Fully Connected (FC) approach. It receives input from the previous layer in the form of a vector created from the flattened feature maps. The output of the FC layer serves as the final output of the network, as depicted in figure 28.

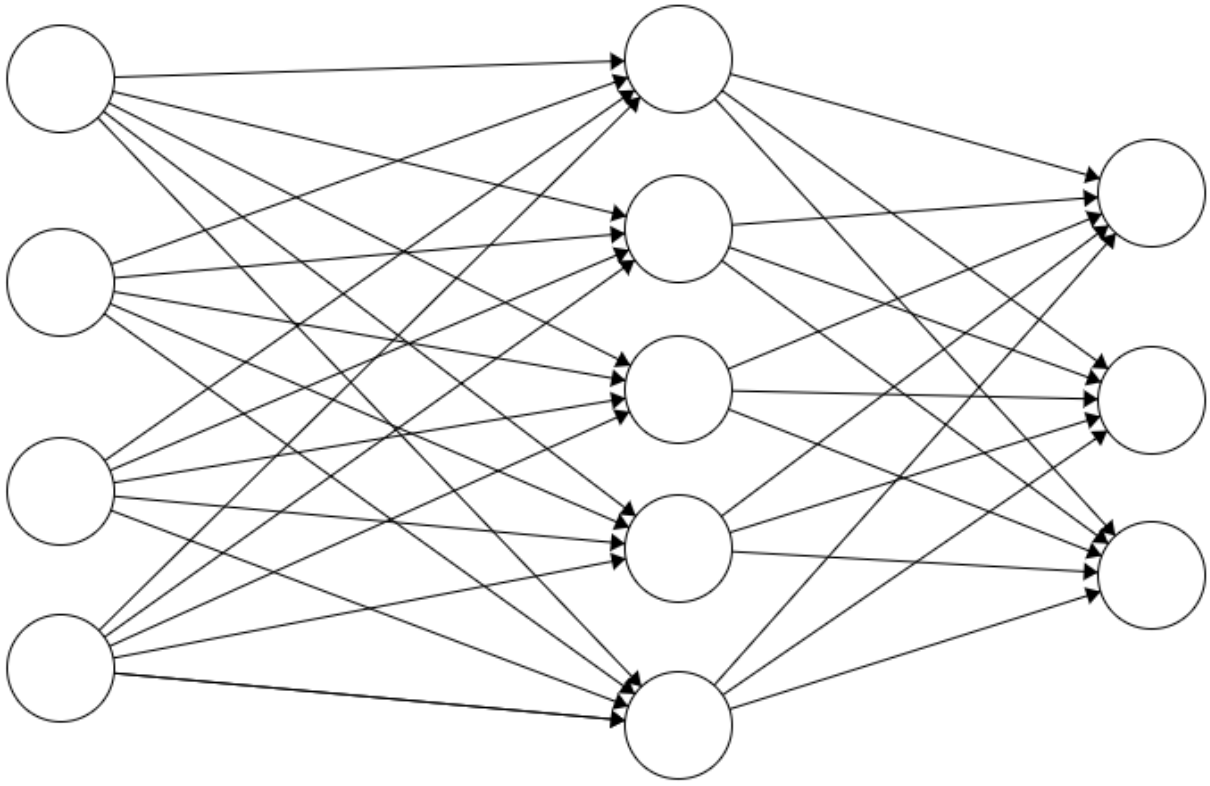

Figure 28: Fully connected layer.

3.1.6. Regularization

The primary issue associated with obtaining a well-behaved generalization in deep learning models is overfitting. Overfitting occurs when the model performs exceptionally well on the training data but fails to perform well on test data, which is unseen data and will be discussed in more detail in a later section. On the other hand, underfitting occurs when the model does not learn enough from the training data, while a "just-fitted" model performs well both on the training and testing data. These three types of models are illustrated in figure 29. To avoid overfitting, various intuitive concepts are used for regularization. We're going to focus on "dropout regularization".

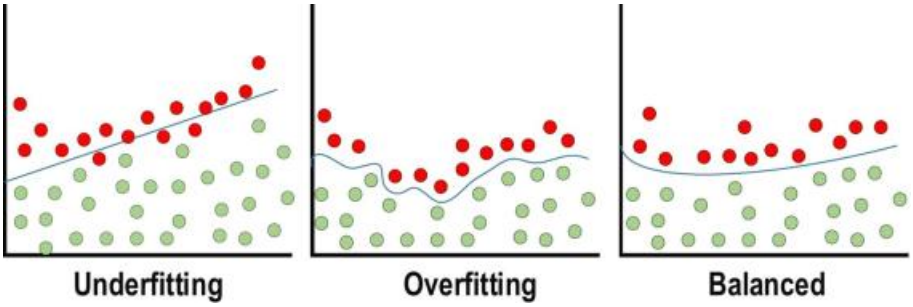

Figure 29: Over-fitting and under-fitting issues.

Dropout

Dropout (figure 30) is a commonly used technique for improving generalization in neural networks. During each training epoch, neurons are randomly dropped from the network (their activations are set to 0), which distributes the feature selection power equally across the entire group of neurons and forces the model to learn independent features. The dropped neuron is not included in the back-propagation or forward-propagation process during training. In contrast, during the testing process, the full network is used to make predictions. For example, a dropout layer with probability p = 0.2 (or keep probability = 0.8) during the forward propagation (training) means that from the previous layer 20% of the nodes would be dropped. [12]

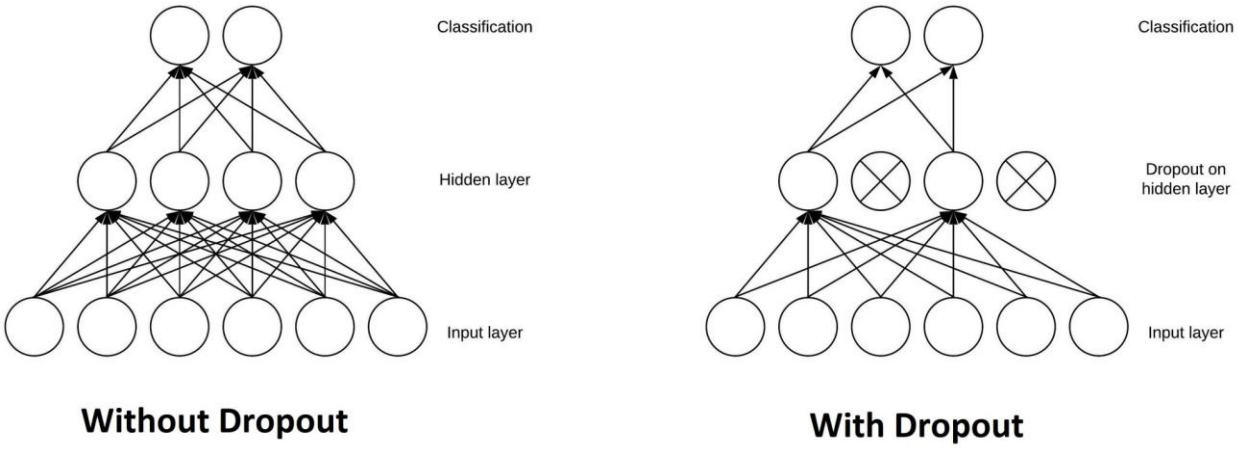

Figure 30: illustration of dropout technique.

3.1.7. Optimizer

The neural network learning process includes selecting a learning algorithm (optimizer) and using enhancements like AdaDelta, Adagrad, and momentum. Loss functions are used in supervised learning algorithms to minimize the error between actual and predicted output. Gradient-based learning techniques are commonly used for neural networks, where network parameters are updated in each training epoch to minimize error. The learning rate determines the step size of parameter updating and should be selected wisely as a hyper-parameter. In our project we're using the "ADAM" optimizer.

ADAM optimizer

Adaptive Moment Estimation (ADAm) is a widely used optimizer in deep learning. It represents the latest trend in optimization and employs a second-order derivative represented by the Hessian matrix. ADAM is specifically designed for training deep neural networks and has two advantages: it is more memory efficient and requires less computational power. ADAM calculates an adaptive learning rate for each parameter in the model. It combines the benefits of both Momentum and Root Mean Square Propagation (RMSprop) optimization techniques. It scales the learning rate using squared gradients as in RMSprop and uses the moving average of the gradient similar to Momentum.

3.1.8. What is LSTM

LSTMs are deeply illustrated in a recent work in [8]. LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neural network (RNN) that is designed to overcome the vanishing gradient problem that can occur in traditional RNNs. The vanishing gradient problem occurs when gradients become too small to effectively update the weights of the network during training, leading to poor performance.

LSTMs solve this problem by introducing a gating mechanism that allows the network to selectively remember or forget information at different time steps. LSTM networks are widely used in natural language processing, speech recognition, and other applications that involve sequential data.

3.1.8.1. LSTM Cell Architecture

The basic architecture of an LSTM cell, shown in figure 31, consists of three gates (input, output, and forget) and a memory cell. The input gate controls how much new information is allowed into the memory cell, while the forget gate controls how much old information is allowed to remain in the memory cell. The output gate determines how much of the memory cell's contents should be output at each time step. LSTM cells usually use the tanh activation function (figure 32).

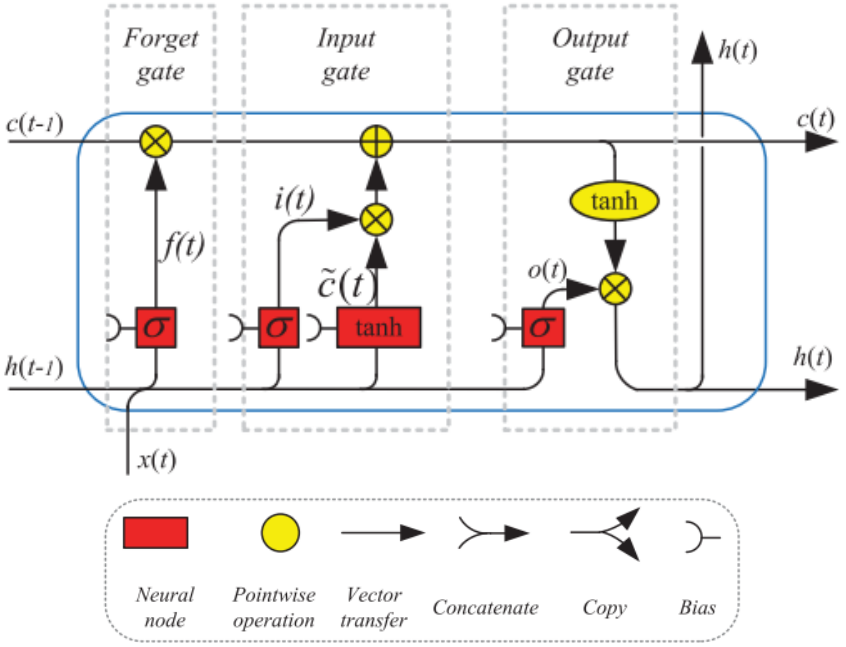

Figure 31: Architecture of LSTM with a forget gate.

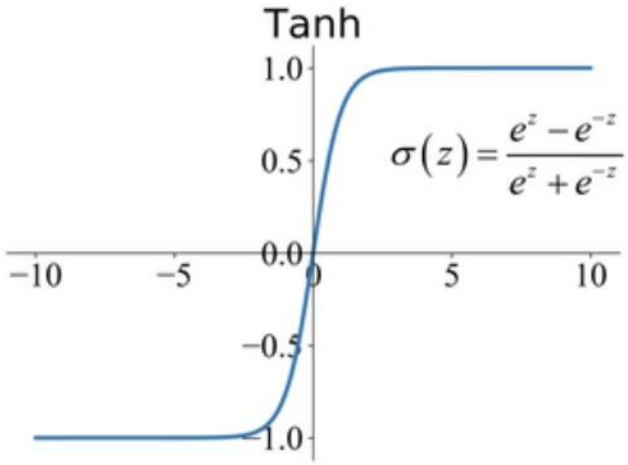

Figure 32: Tanh function

The LSTM inputs and outputs and gates are illustrated in [8].

3.1.8.2. Input and Output

In an LSTM cell, there are three input vectors: the cell state (ct-1), the hidden state (ht-1), which are generated by the LSTM in the previous time step (t-1), and the input vector x(t) from an external source. By regulating the flow of information and transforming the values of the cell and hidden state vectors, the LSTM controls the internal information flow through its gates. These transformed vectors will be part of the input set for the next time step (t+1). The aim of this information flow control is to ensure that the cell state acts as a long-term memory, while the hidden state acts as a short-term memory. To update the long-term memory (cell state, C), the LSTM uses both recent past information (the short-term memory, h) and new information from the outside (the input vector, x). The long-term memory is then used to update the short-term memory (hidden state, h) and determine the output of the LSTM unit at time step t. This output is what the LSTM provides to the external environment to perform a specific task.

3.1.8.3. Gates

The forget gate, input gate, and output gate in an LSTM are responsible for selecting information and creating selector vectors, which are vectors with values that are close to zero or one. These selector vectors are designed to be element-wise multiplied by a vector of the same size. In this multiplication process, a position where the selector vector has a value of zero eliminates the corresponding information in the other vector, whereas a position where the selector vector has a value of one does not alter the information in the other vector. All three gates are neural networks that utilize the sigmoid function as their activation function in the output layer to produce vectors with values that are close to zero or one. Additionally, all three gates take in an input vector (X) and the hidden state vector from the previous time step (h_{t-1}), which are concatenated together into a single vector that serves as the input for all three gates.

3.1.8.3.1. Forget Gate

At each time step t, an LSTM takes in an input vector x(t), as well as the hidden state ht−1 and cell state ct−1 vectors from the previous time step (t-1). The forget gate determines which information to remove from the cell state vector from time step t-1 based on the input vector x(t) and ht−1 vector, producing a selector vector. The selector vector is element-wise multiplied with the cell state vector received as input, eliminating information where the selector vector has a value of zero and leaving information unchanged where the selector vector has a value of one.

3.1.8.3.2. Input Gate and Candidate Memory

To update the cell state in an LSTM, two independent neural networks called the candidate memory and input gate use concatenated input vectors X_[t] and H_[t−1] to generate a candidate vector and selector vector, respectively. The candidate vector's values are normalized to -1 and 1 using the hyperbolic tangent function. The selector vector and candidate vector are element-wise multiplied, eliminating information where the selector vector has a value of zero and leaving information unchanged where the selector vector has a value of one. The resulting vector is added

to the cell state vector to add new information. The updated cell state is then used by the output gate and passed into the input set used by the LSTM unit in the next time step (t+1).

3.1.8.3.3. Output Gate

The output gate determines the hidden state value that is outputted by the LSTM at time step t and received by the LSTM in the next time step (t+1) input set. The output generation process involves multiplying a selector vector with a candidate vector. The candidate vector is obtained by applying the hyperbolic tangent function to the cell state vector, normalizing the vector values within a range of -1 to 1. The selector vector is generated by the output gate using the sigmoid function as the activation function of the output neurons. The multiplication between the selector vector and the candidate vector eliminates information in the candidate vector where the selector vector has a value of zero and leaves information unchanged where the selector vector has a value of one.

3.1.8.4. LSTM Network Architecture

The most common and simple method to increase the capacity and depth of an LSTM network is to stack LSTM layers, creating a multi-layer fully connected structure. A block of three recurrent layers is illustrated in figure 33. Then we unroll this schematic along the time dimension in figure 34, where we assume that the sequence length is 4. In the depth dimension of the stacked LSTM network, the output of the $(L-1)$th LSTM layer at time t is h_t^{L-1}.This output is treated as the input x_t^L of the Lth layer. These output-input connections are the only relation between two adjacent layers. In the time dimension, the recurrent connections are only within one layer. This means that the cell state and hidden state from the previous time step of the same layer are used as inputs along with the input vector for the current time step. The LSTM unit processes these inputs and produces an output that is passed to the next time step. Due to its simplicity and efficiency, the stacked LSTM network has been widely used for various applications. For instance, it has been adopted to solve vehicle-to-vehicle communication problems, and the stacked LSTM-based regression model has shown higher efficiency than logistic regression. Additionally, in an English-to-French translation task, reversing the order of source words improved the performance, while a deep stacked LSTM network was used to predict the intent of pedestrians in traffic scenarios and outperformed a methodology relying on hand-crafted features.

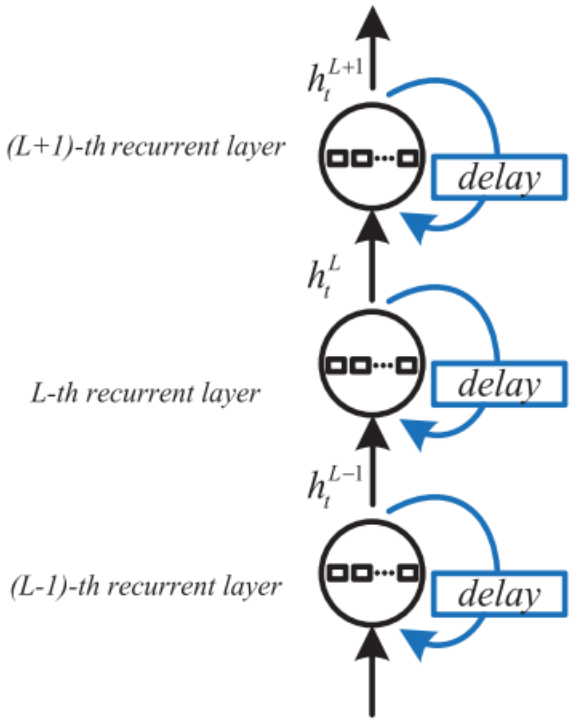

Figure 33: The stacked LSTM network.

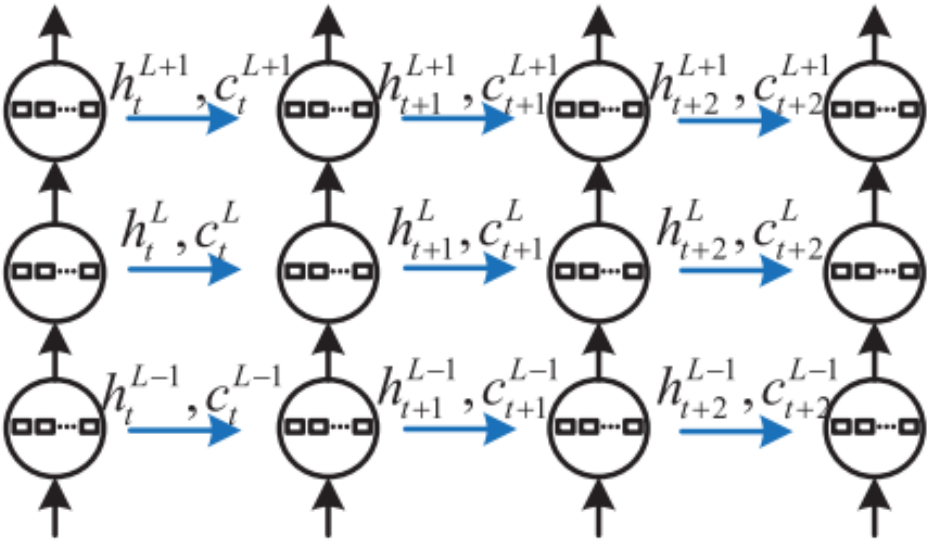

Figure 34: An unrolled stacked LSTM network.

3.2.Performance Metrics

The evaluation metrics utilized in Deep Learning (DL) tasks are crucial for optimizing the classifier. These metrics are used in two main stages of a data classification procedure: training and testing. During the training stage, the evaluation metric is employed to optimize the classification algorithm. It serves as a discriminator, helping to select the most optimized solution that can provide highly accurate predictions for upcoming evaluations related to a specific classifier. In the testing stage, the evaluation metric is used to measure the efficiency of the created classifier. It acts as an evaluator, assessing the performance of the model using hidden data, which refers to a set of data that is not used during the training phase of a model, but is instead reserved for testing or evaluating the performance of the model. The number of successfully classified negative and positive instances is defined as True Negative (TN) and True positive (TP), respectively, while the number of misclassified positive and negative instances is defined as False Negative (FN) and False Positive (FP), respectively.

3.2.1. Accuracy

Accuracy measures the ratio of correctly predicted classes to the total number of samples evaluated. It is computed as the sum of true positives and true negatives divided by the sum of true positives, true negatives, false positives, and false negatives as given in equation [9]. It provides a simple and intuitive measure of a model's overall performance. However, it may not be the most suitable metric in all cases, especially when dealing with imbalanced datasets, where one class has much fewer samples than the other.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (9)$$

3.2.2. Precision

Precision measures the proportion of correctly predicted positive patterns among all the predicted positive patterns. It is computed as the ratio of true positives (TP) to the sum of true positives and false positives (FP) as given in equation [10]. Precision is a useful metric when the focus is on minimizing false positives, that is, when it is important to ensure that the positive predictions made by the model are highly accurate and reliable. However, precision alone may not provide a complete picture of a model's performance, especially when dealing with imbalanced datasets.

$$Precision = \frac{TP}{TP+FP} \qquad (10)$$

3.2.3. Recall

Sensitivity, also known as recall or true positive rate, measures the fraction of positive patterns that are correctly classified by the model. It is computed as the ratio of true positives to the sum of true positives and false negatives as given in equation [11]. Recall is a useful metric in cases where the focus is on minimizing false negatives, that is, when it is important to ensure that all positive patterns are correctly classified by the model. However, recall alone may not provide a complete picture of a model's performance, especially when dealing with imbalanced datasets.

$$Recall = \frac{TP}{TP+FN} \quad (11)$$

3.2.4. F1-score

F1-score (equation [12]) calculates the harmonic mean of the precision and recall rates of a model. It is a single number that provides a balanced measure of a model's performance by combining both precision and recall and is often used in classification tasks where the classes are imbalanced. The harmonic mean is used to combine precision and recall because it gives more weight to lower values. This means that if either precision or recall is low, the F1 score will be low as well, regardless of the other metric. This makes the F1 score a more robust metric than precision or recall alone, especially when dealing with imbalanced datasets.

$$F1_{score} = 2 * \frac{Precision \times Recall}{Precision + Recall} \qquad (12)$$

3.3. The Proposed Deep Learning Architectures

In this study, two models are implemented to determine the presence of jammer and identification of the attack type: (1) A CNN-based model, and (2) An LSTM-based model.

3.3.1. I/Q Sample Representation in the CNN Model

I-Q samples can be represented as a two-dimensional greyscale image (virtual image), where one axis represents the I component and the other axis represents the Q component. The CNN can then be trained on a dataset of labeled images, with the labels indicating whether the image represents a normal signal or a jamming signal. During training, the CNN learns to extract features from the image that are relevant to distinguishing between normal and jamming signals. Once the CNN is trained, it can be used to classify new images as either normal or jamming signals. This approach has been shown to be effective in detecting and classifying a wide range of jamming techniques in wireless communication systems. [13]

3.3.2. I/Q Sample Representation in the LSTM Model

The I/Q samples have been used as features in the literature in [9], where they used Orthogonal Frequency Division Multiplexing (OFDM) with QPSK modulation as input data to classify the presence and type of the jamming signal. LSTM can be applied to I-Q samples data in a manner similar to other forms of sequential data, including time series and natural language. In this case, the I-Q samples can be viewed as a sequence of data, with each sample representing a time step. By utilizing the LSTM algorithm on this sequence, the model can be trained to identify patterns and connections between the I-Q samples.

3.3.3. Data shape

In both models, we split the sequence of data into n frames with frame size 128 shown in figure 35. Note: the frame size = 128 gave the maximum performance after tuning.

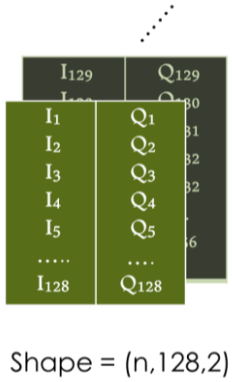

Figure 35: Frame Format.

3.4. Applying The Models on the Datasets

3.4.1. USRP Dataset Models

3.4.1.1. Data preprocessing

Outliers are data points that deviate significantly from the majority of the data and can negatively affect the performance of machine learning models or statistical analyses. For data preprocessing, we removed the 1st and 99th percentiles of the I-Q components of the dataset. This preliminary filtering process is aimed at identifying and removing inconsistent I-Q samples that may cause bias in the analysis or model training. By eliminating these outliers, the remaining samples are more representative of the true distribution of the data, which leads to improved accuracy and performance for the models.

Figures 36 and 37 show the data distribution of the in-phase and quadrature components in scenario W1 before and after removing outliers. We can see the range has improved significantly after removing outliers.

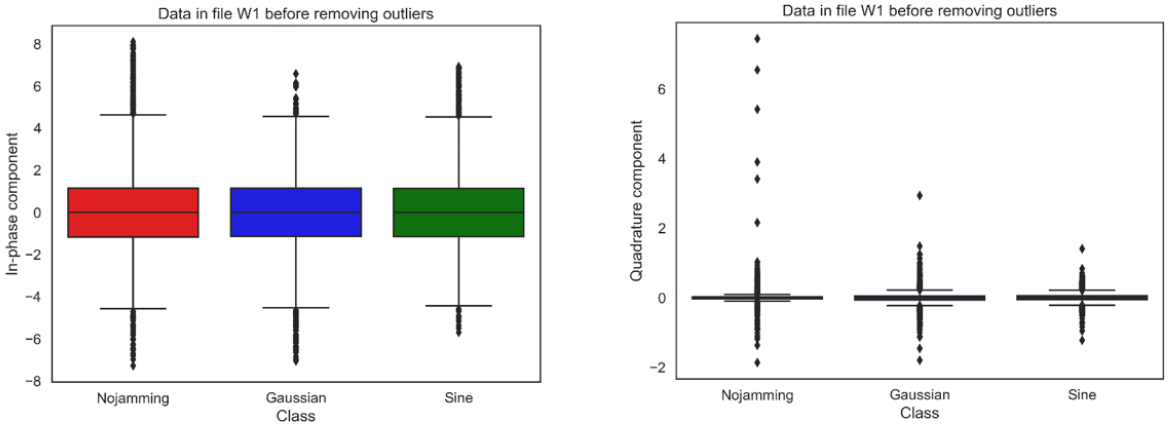

Figure 36: I/Q components of W1 before removing outliers

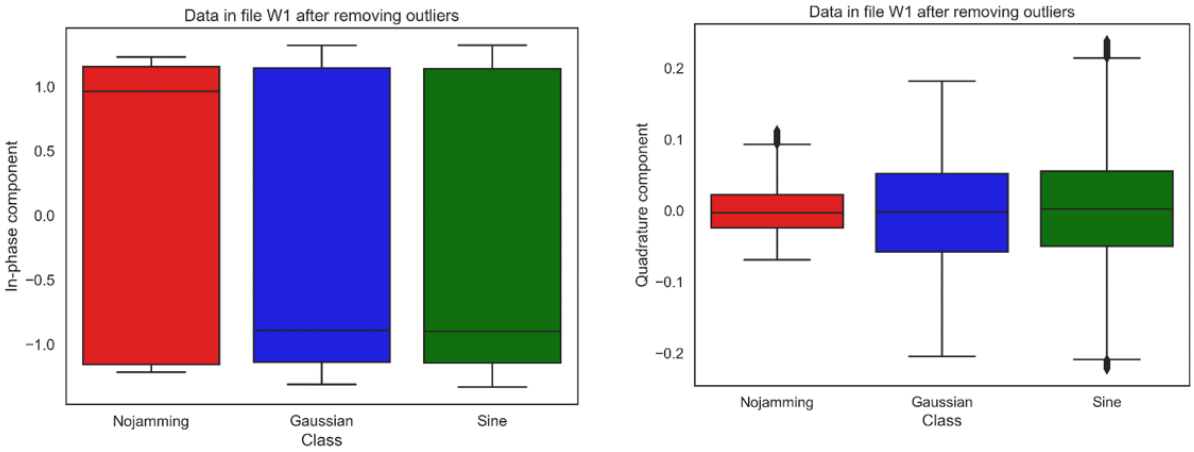

Figure 37: I/Q components of W1 before removing outliers

3.4.1.2. Data Splitting Ratios for Training, Testing and Validation

Data splitting is an essential step in training deep learning models, including CNNs and LSTMs. The goal of data splitting is to evaluate the performance of the model on data that it has not seen during training and to tune the hyper-parameters of the model to achieve better performance. The typical split involves dividing the available data into three subsets: the training set, the validation set, and the test set. The training set is used to train the model. The validation set is used to evaluate the model's performance during training and to tune the hyper-parameters, and the test set is used to evaluate the final performance of the model after training and hyper-parameter tuning. The recommended split ratio between these subsets can vary depending on the size of the dataset and the complexity of the problem. Also, it's important to note that the test set should only be used once, after the model has been fully trained and tuned, to avoid overfitting to the test set.

In our problem we used 80% for the training set, 10% for the validation set, and 10% for the test set. Before splitting we randomly shuffled the data to avoid any bias in the data distribution across the subsets.

3.4.1.3. Training Approaches

First Approach: Train on One, Test on the Rest

Due to lack of computational resources, as the available data is about 110 Gigabytes, we tried to consider only one scenario to train the models, i.e., one value for Jamming Power, one value for distance between the transmitter and the receiver and using one Jamming Device. We chose a medium relative jamming power (0.6) and a medium distance (10 m) for the training scenario. However, when the models were tested on the rest of the scenarios, results were not as expected, as it could not perform well on any jamming power.

Second Approach: Diverse Dataset

We used $5 \cdot 10^5$ (500,000) data samples of each class (no jamming, Gaussian, sine) in each file and stored it in a csv file along with jammer ID, jamming power, distance, labels as shown in figure 38. This approach enabled us to capture all scenarios without needing high computational power as each file contains more than **150 million** data points for each class and there are 31 files (scenarios).

Results using this approach were comprehensive as the model was able to classify the presence of jamming signal at high jamming power which is our target.

```
RangeIndex: 46500000 entries, 0 to 46499999
Data columns (total 6 columns):
 #   Column    Dtype
---  ------    -----
 0   I         float64
 1   Q         float64
 2   id        object
 3   power     object
 4   distance  object
 5   labels    object
```

Figure 38: Diverse Dataset

3.4.1.4. CNN Model

We used a CNN model architecture with an input shape of (n, 128,2), where "n" is the number of frames that are used as input into the model, 128 is the frame size, and the 2 is for "I" and "Q". The model's total number of trainable parameters is 2,859. The used model consisted of a 16-unit convolutional layer with a filter size of 3 and ReLU activation; a maxpooling layer; an 8-unit convolutional layer with ReLU activation; a flatten layer; and a 3-unit dense layer with SoftMax activation for the 3 jamming classes. Figure 39 shows the architecture.

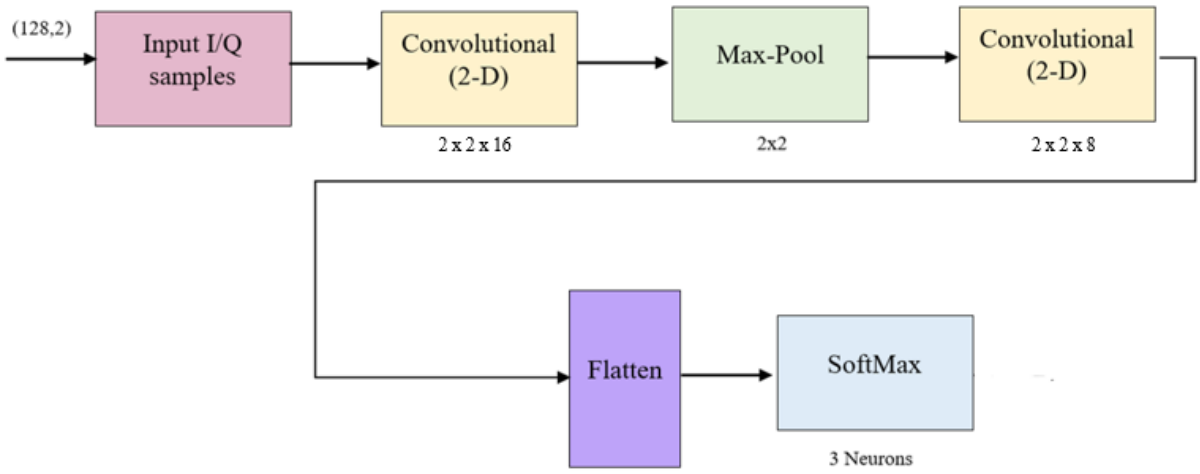

Figure 39: CNN Architecture Trained on USRP Dataset.

3.4.1.5. LSTM Architecture

We used an LSTM model architecture with the same input shape as the CNN model. The model's total number of trainable parameters is 8,259. The used model consisted of a 32-unit LSTM layer; a 16-unit LSTM layer; a 32-unit dense layer with ReLU activation; a dropout layer; and a 3-unit dense layer with SoftMax activation for the 3 jamming classes. Figure 40 shows the architecture.

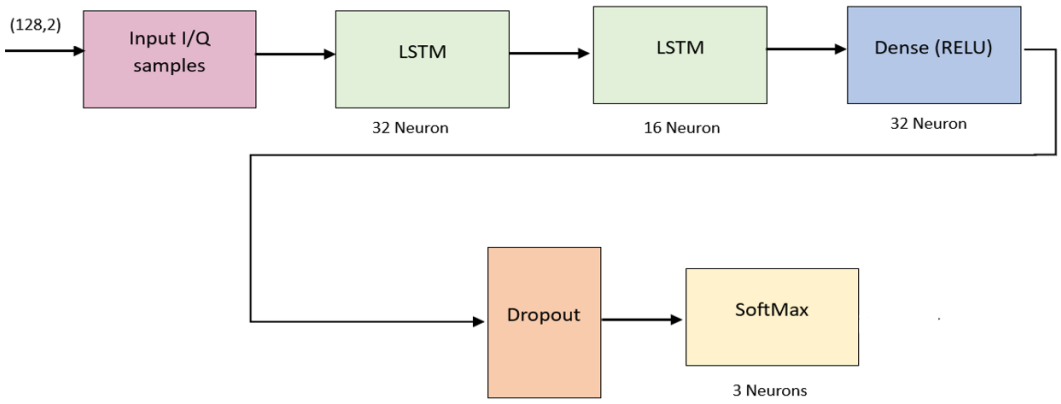

Figure 40: LSTM Architecture Trained on USRP Dataset

3.4.2. USRP Dataset Models Results

The performance of the CNN and LSTM models for wireless jamming detection was evaluated by measuring their mean test accuracies as well as the standard deviation across different jamming power values. The following tables summarize the mean accuracies and standard deviations obtained for both models at various jamming power levels.

3.4.2.1. Model Accuracy with respect to Jamming Power

Table 2: CNN Model Accuracy w.r.t Jamming Power

CNN Accuracy w.r.t Jamming Power								
Jamming Power	0.1	0.2	0.3	0.4	0.5	**0.6**	**0.7**	**0.8**
Accuracy Mean	0.662	0.712	0.756	0.816	0.896	0.932	0.987	0.987
Accuracy Standard Deviation	0.199	0.230	0.238	0.059	0.101	0.077	0.000	0.002

Table 3:LSTM Model Accuracy w.r.t Jamming Power

LSTM Accuracy w.r.t Jamming Power								
Jamming Power	0.1	0.2	0.3	0.4	0.5	**0.6**	**0.7**	**0.8**
Accuracy Mean	0.673	0.691	0.706	0.909	0.923	0.968	0.999	1.000
Accuracy Standard Deviation	0.254	0.286	0.289	0.031	0.093	0.051	0.000	0.002

As seen in Table [2], the CNN model achieved mean accuracies ranging from 66.2% to 98.7% across the different jamming power levels. As the jamming power increased, the model's accuracy generally improved. Notably, at a relative jamming power of greater than 0.6, the CNN model consistently achieved high accuracies of 98.7%. And since the BER isn't significantly affected below that range [13] as shown in figure 41, the model is to be considered capable of achieving "early detection". The standard deviations of the accuracies varied, with the values decreasing as the relative jamming power increased. At a relative jamming power of 0.7 and 0.8, the standard deviations were extremely low, indicating consistent and reliable performance.

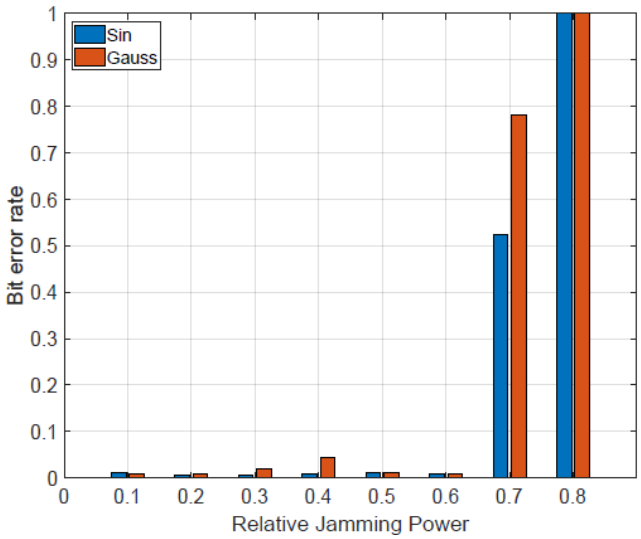

Figure 41: BER vs. Relative Jamming Power [13]

Table [3], presents the results for the LSTM model. The mean accuracies for the LSTM model ranged from 67.3% to 100%. Similar to the CNN model, the accuracy improved with increasing relative jamming power. Notably, at a relative jamming power of 0.7 and 0.8, the LSTM model achieved near-perfect accuracies of 99.9% and 100%, respectively. The standard deviations of the accuracies were generally higher than those of the CNN model but remained relatively low overall. At a relative jamming power of 0.7 and 0.8, the standard deviations were exceptionally low, indicating consistent and highly reliable performance.

Figure (42) shows the mean accuracies plotted as error-bar plots against the relative jamming power with the standard deviation presented as the error bars.

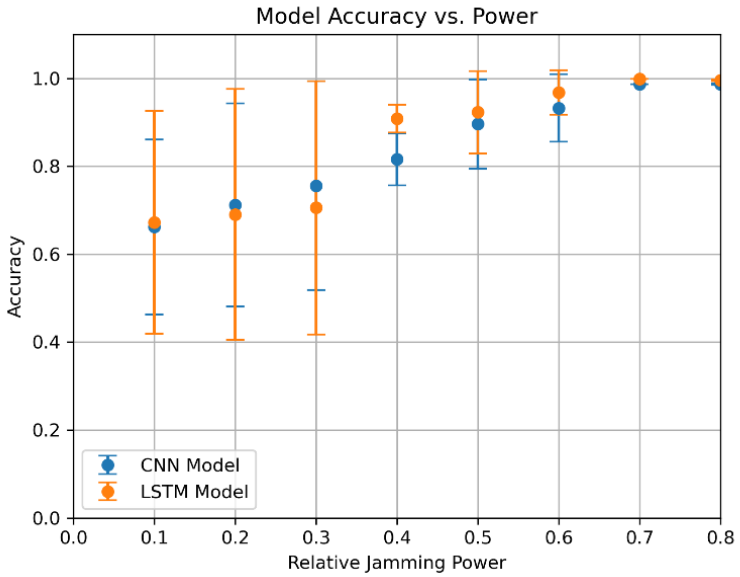

Figure 42: USRP Models Accuracy vs. Jamming Power

3.4.2.2. Model Accuracy with respect to Tx-Rx Distance

The performance of the models was further evaluated by measuring their mean test accuracies across different distances between the transmitter (Tx) and the receiver (Rx). The following tables summarize the mean accuracies obtained for both models at various distances.

Table 4: CNN Model Accuracy w.r.t Tx-Rx Distance

CNN Accuracy w.r.t Tx-Rx Distance								
Distance between Tx and Rx (m)	3	5	7	10	13	16	19	21
Accuracy Mean	0.970	0.823	0.698	0.874	0.969	0.910	0.923	0.738

Table 5: LSTM Model Accuracy w.r.t Tx-Rx Distance

LSTM Accuracy w.r.t Tx-Rx Distance								
Distance between Tx and Rx (m)	3	5	7	10	13	16	19	21
Accuracy Mean	0.985	0.926	0.791	0.890	0.969	0.906	0.990	0.721

In Table [4], the CNN model achieved overall high mean accuracies ranging from across the different distances between the transmitter and the receiver. Generally, the accuracy of the model doesn't seem to depend on the Tx-Rx distance, and the variations in accuracy are due to dataset imbalances (as some distances are used with lower jamming powers, for which the models don't have high accuracies). It must be noted that the measurements stop at 21 meters due to the fact that the communication link is not reliable after that anyway regardless of whether there's jamming or not.

Table [5], presents the results for the LSTM model. Similar to the CNN model, the accuracy doesn't depend on the distance, and dataset imbalances cause minimal variations.

These results demonstrate that both the CNN and LSTM models are effective in detecting and classifying jamming techniques at different distances between the transmitter and the receiver. These findings highlight the models' ability to adapt to varying distances between the transmitter and the receiver, ensuring reliable jamming detection in real-world scenarios. The insights gained from these results can guide the selection and optimization of deep learning models for wireless jamming detection, considering the specific distance requirements of different applications and deployment scenarios.

Figure (43) shows the mean accuracies plotted as bar plots against the Tx-Rx distance.

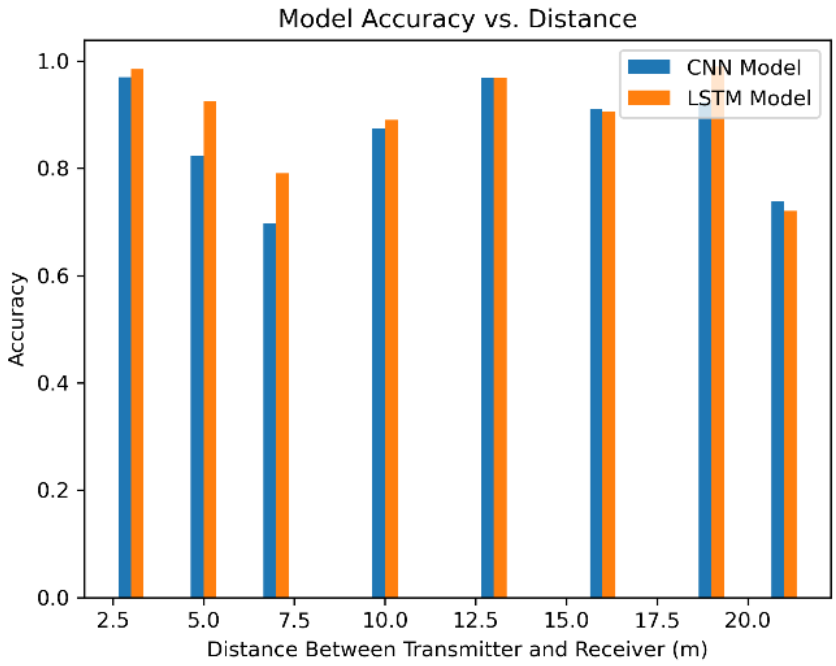

Figure 43: USRP Model Accuracy vs. Tx-Rx Distance

3.4.2.3. Model Accuracy with respect to Jamming Device

The performance of the CNN and LSTM models for wireless jamming detection was further evaluated by measuring their mean test accuracies across different jamming devices. Each jamming device was assigned a unique Jamming Device ID, allowing us to differentiate them. The following tables summarize the mean accuracies.

Table 6: Table 5: CNN Model Accuracy w.r.t Jamming Device

CNN Accuracy w.r.t Jamming Device					
Jamming Device ID	4	5	6	7	8
Accuracy Mean	0.916	0.791	0.940	0.988	0.691

Table 7: LSTM Model Accuracy w.r.t Jamming Device

LSTM Accuracy w.r.t Jamming Device					
Jamming Device ID	4	5	6	7	8
Accuracy Mean	0.946	0.794	0.960	0.995	0.721

In table [6], the CNN model achieved overall high mean accuracies across different jamming devices. The performance of the CNN model seems to not depend on the jamming device used, and the slight variations are due to dataset imbalances (not all jamming devices are represented equally in the dataset).

Table [7], presents similar results for the LSTM model.

Figure (44) shows the mean accuracies plotted as bar plots against the jamming device used.

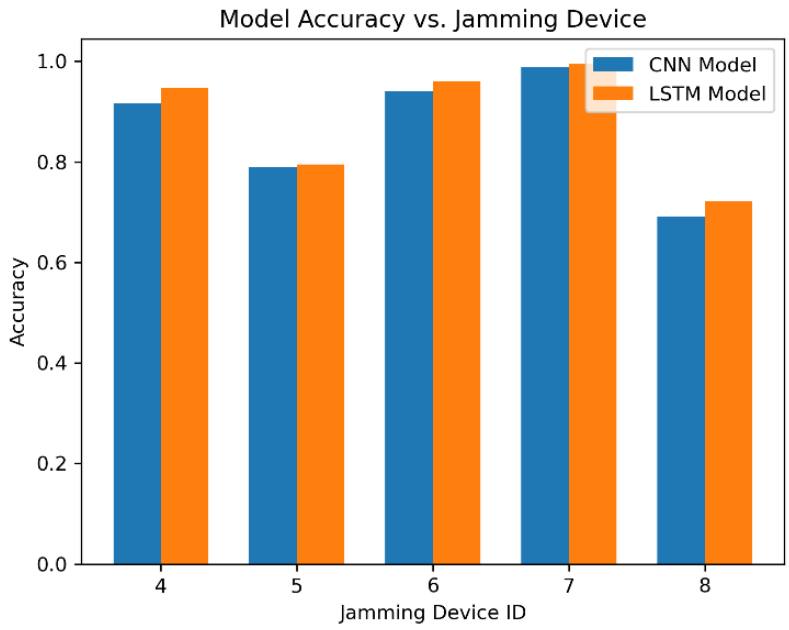

Figure 44: USRP Model Accuracies vs Jamming Device used

These results demonstrate that both the CNN and LSTM models are capable of detecting and classifying jamming techniques from different jamming devices. The CNN model exhibited slightly lower mean accuracies compared to the LSTM model, but both models demonstrated strong performance overall. It must be noted that with more training these accuracies can be made constantly above 95% across the board (which will be demonstrated in the generated dataset part).

3.4.2.4. Model Confusion Matrices

A confusion matrix is a table that summarizes the performance of a classification model. It is often used in machine learning and statistics to evaluate the accuracy of a predictive model. The matrix compares the predicted labels of the model against the actual labels of the data set being tested.

Figures 45 and 46 show the confusion matrices of the USRP dataset-trained CNN and LSTM models respectively.

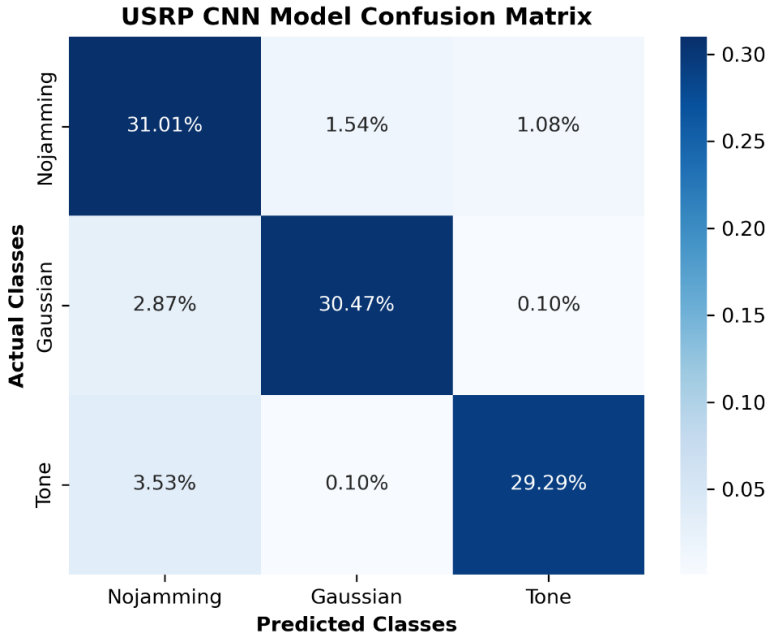

Figure 45: USRP CNN Model Confusion Matrix

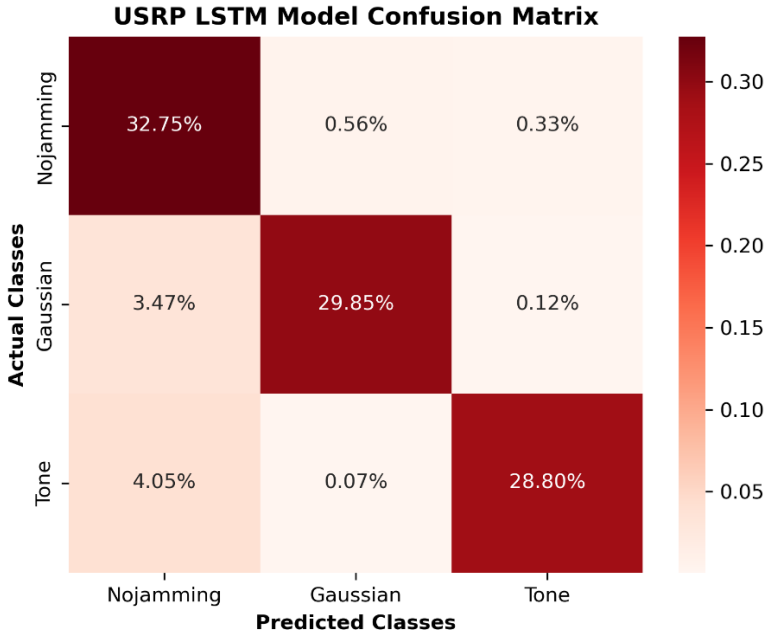

Figure 46: USRP LSTM Model Confusion Matrix

3.4.2.5. Model Precision, Recall, and F1-score

Tables 8 and 9 show the precision, recall, and f1-score metrics that were calculated for the USRP dataset-trained CNN and LSTM models respectively.

Table 8: CNN Model Metrics

CNN Model			
Class	Precision	Recall	F1-Score
Nojamming	0.8289	0.9218	0.8729
Gaussian	0.9487	0.9112	0.9296
Tone	0.9611	0.8896	0.9239

Table 9: LSTM Model Metrics

LSTM Model			
Class	Precision	Recall	F1-Score
Nojamming	0.8132	0.9736	0.8862
Gaussian	0.9793	0.8926	0.9340
Tone	0.9846	0.8748	0.9264

As shown, the metrics are mostly very high which means that the models are capable of differentiating between the different jamming scenarios well.

3.4.3. A Defect in the USRP Dataset

After training and testing our models on the USRP data and the results were very satisfying, it was necessary to test these models on other data with different channel conditions to see if they generalize well. As expected, however, the models don't generalize well on datasets with worse channel conditions.

The reason for that is that the USRP dataset has high SNRs which made the model unable to recognize the data when lower SNRs were used, which is why it was necessary to generate our own dataset with variable values of SNR and variable values of jamming powers to make the model able to recognize various scenarios of data even when channel conditions aren't ideal. We also transmitted this new dataset using the Blade RF X40 board to introduce other kinds of non-idealities (which will be discussed later on) to simulate some real-life imperfections.

3.4.4. Generated Dataset Models

In this section, we present the models that were trained on the generated dataset, as well as a comparison of different architectures used for these models. The goal was to identify the optimal number of parameters while achieving high metrics for the classification problem in the three classes. To determine the impact of parameter variations on model performance, we systematically increased and decreased the number of parameters until we reached certain accuracy thresholds. The results of the performance evaluation were recorded and analyzed, leading to the identification of the best-performing model configurations.

3.4.4.1. CNN Models

The general architecture of the CNN model consisted of one or more Conv2D layers with ReLU activation, 2x2 filters, and "same" padding; a max pooling layer; a dropout rate of 0.2; an optional dense layer with ReLU activation; and a final dense layer with SoftMax activation [11].

- The reason for turning off the biases in CNN model:

A bias in a neural network unit refers to the "free term" that is added to the weighted sum of the unit's inputs. It's a trainable parameter just like the weights. Biases can be turned off in neural network layers if needed. One of the reasons why they can be chosen to be turned off is for model quantization.

Whether to turn off biases before quantizing weights in a Convolutional Neural Network (CNN) depends on the specific scenario and quantization scheme being used.

In some cases, turning off bias before quantization can help improve the accuracy of the quantized model, as biases are often quantized using fewer bits than weights and can therefore lead to greater quantization error. However, turning off bias can also lead to reduced model accuracy, especially if the quantization scheme being used relies on the relative magnitudes of weights and biases to perform accurate quantization.

Therefore, it is recommended to experiment with both options (i.e., with and without bias) and determine the best approach based on the specific requirements of the application and the characteristics of the model. And that is what we have done. We tried both turning on and turning off biases in the CNN model while training before quantizing its weights then we tested both versions and the result was that the version with biases was less in performance than the other, so we kept them off.

Table 10: CNN Model Performance Evaluation

Filter sizes = 2x2						
	Number of Parameters	Test Accuracy	Class	Precision	Recall	F1-Score
Extra Dense Layer	4696	99.40%	Nojamming	0.9827	1.0000	0.9913
			Gaussian	1.0000	1.0000	1.0000
			Tone	1.0000	0.9828	0.9913
	2636	98.40%	Nojamming	0.9541	0.9998	0.9764
			Gaussian	0.9998	0.9996	0.9997
			Tone	1.0000	0.9537	0.9763
	1356	97.13%	Nojamming	0.9236	0.9972	0.9590
			Gaussian	0.9990	0.9982	0.9986
			Tone	0.9980	0.9181	0.9564
Without Extra Dense Layer	4160	98.50%	Nojamming	0.9573	0.9984	0.9774
			Gaussian	1.0000	1.0000	1.0000
			Tone	0.9983	0.9562	0.9768
	2112	98.30%	Nojamming	0.9524	0.9998	0.9755
			Gaussian	1.0000	1.0000	1.0000
			Tone	0.9998	0.9506	0.9746
	784	91.80%	Nojamming	0.8453	0.9214	0.8817
			Gaussian	0.9998	0.9992	0.9995
			Tone	0.9141	0.8325	0.8714

Table [10] shows the performance evaluation of the CNN model. The best model achieved 99.4% accuracy with 4,696 parameters, including the optional dense layer. Additionally, when removing the optional dense layer, we observed a slight decrease in accuracy to 98.5% with 4160 parameters.

To further optimize the model, we explored reducing the number of parameters while maintaining acceptable accuracy levels. With the optional dense layer, a reduction to 2,636 parameters resulted in 98.4% accuracy. Without the optional dense layer, we achieved 98.3% accuracy with only 2112 parameters.

The precision, recall, and f1-score metrics were also calculated to check whether the model has biases towards certain classes or not. We observe that the model doesn't have a bias towards a certain class, as all precisions, recalls, and F1-scores are very high.

It is worth noting that while the number of parameters was reduced, the metrics remained relatively high, showcasing the efficiency of these configurations. These results demonstrate the trade-off between model complexity (as represented by the number of parameters) and performance. By carefully selecting the number of parameters, we were able to strike a balance between model size and accuracy, enabling efficient training and inference for our classification problem.

In further analysis of the CNN model, the 4,696-parameter version (the maximum-performance version) will be used to explore the maximum potential that the model has. In this version, 2 Conv2D layers are used. The first Conv2D layer consists of 16 units. The second one consists of 8 units. The optional dense layer is used, and it consists of 8 units. The classification dense layer consists of 3 units (the same as the number of classes). Figure [47], shows the architecture of the maximum-performance version of the CNN model.

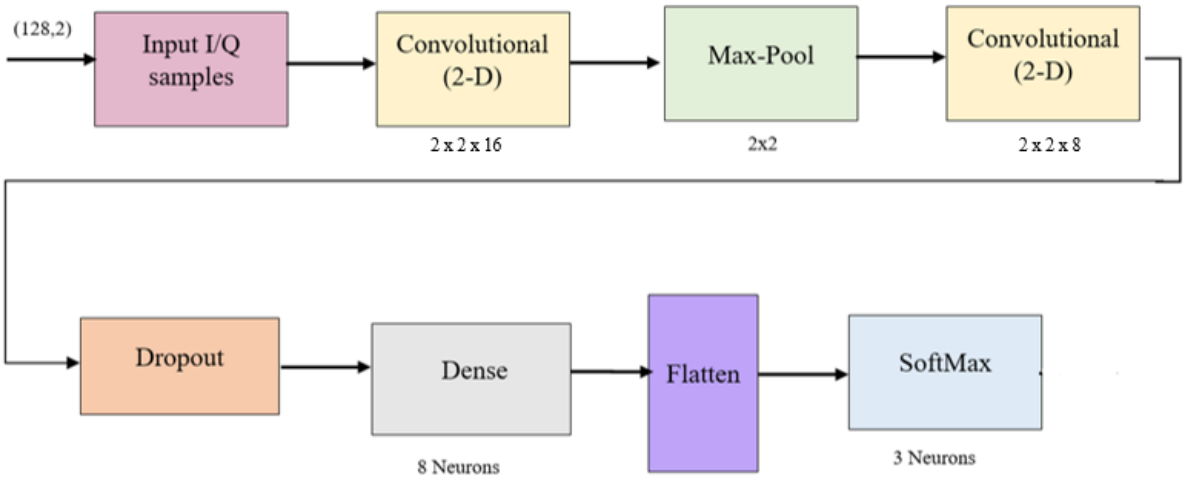

Figure 47:Maximum-performance CNN Model Architecture

Figure 48 shows the confusion matrix of the maximum-performance version of the CNN model.

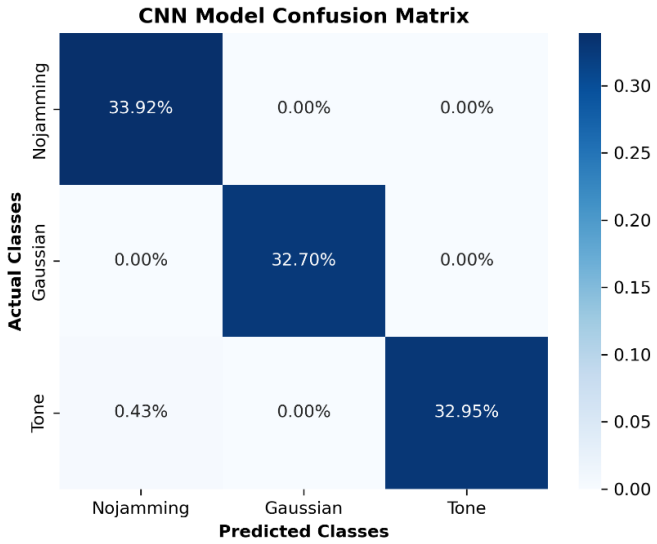

Figure 48: CNN Model (Max Performance) Confusion Matrix

3.4.4.2. LSTM Model

The general architecture of the LSTM model consisted of one LSTM layer with tanh activation; a dense layer with ReLU activation; and another dense layer with SoftMax activation [8].

Table 11: LSTM Model Performance Evaluation

Number of Parameters	Test Accuracy	Class	Precision	Recall	F1-Score
1539	99.30%	Nojamming	0.9795	0.9990	0.9891
		Gaussian	1.0000	1.0000	1.0000
		Tone	0.9990	0.9800	0.9894
451	99.00%	Nojamming	0.9748	0.9968	0.9857
		Gaussian	1.0000	1.0000	1.0000
		Tone	0.9967	0.9741	0.9852
147	93.30%	Nojamming	0.8571	0.9970	0.9218
		Gaussian	0.9998	0.9998	0.9998
		Tone	0.9967	0.8339	0.9081

Table 11 shows the performance evaluation of the LSTM model. The best model achieved 99.3% accuracy with 1539 parameters (maximum performance). Reducing the number of parameters to 451 results in a slight loss in accuracy but a high increase in speed.

Looking at the precision, recall, and F1-score metrics, we can see that also the LSTM model isn't biased towards one class over the other, and that it can accurately differentiate between them.

It's also worth noting that a relatively high accuracy is achieved with the LSTM model with a smaller number of parameters than the corresponding accuracy from the CNN model.

In further analysis of the LSTM model, the 1539-parameter version (the maximum-performance version) will be used to explore the maximum potential that the model has. In this version, the LSTM layer consists of 16 cells; the first dense layer consists of 16 units, and the last dense layer consists of 3 units for the classification task. Figure (49) shows the architecture of the maximum-performance version of the LSTM model.

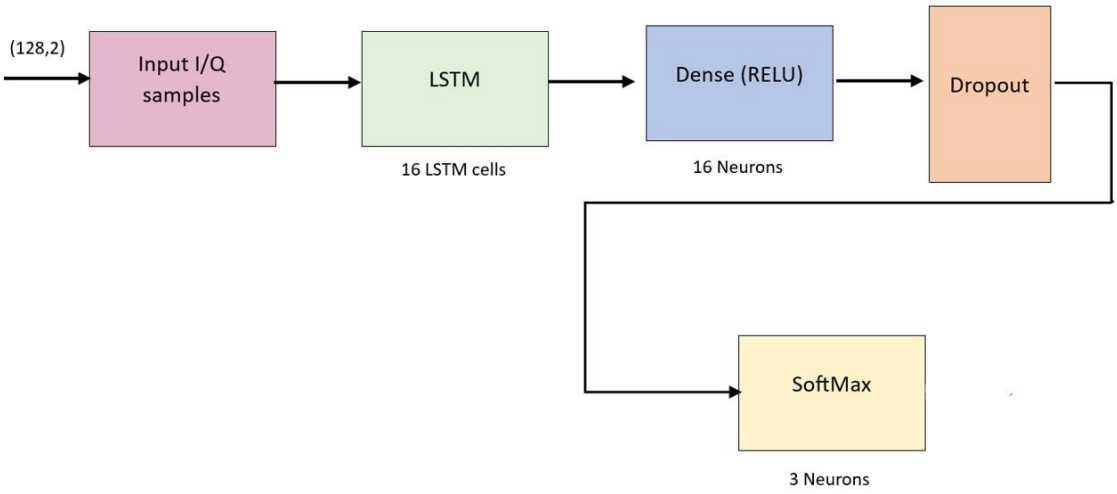

Figure 49: Maximum-performance LSTM Model Architecture

Figure 50 shows the confusion matrix of the maximum-performance version of the LSTM model.

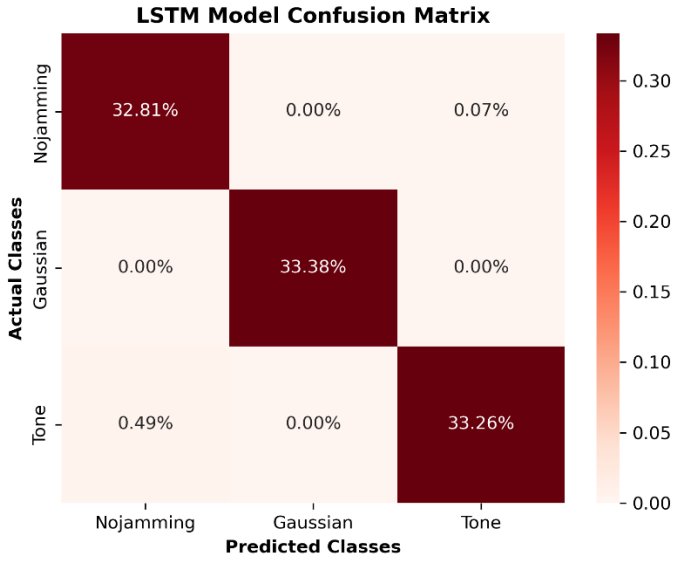

Figure 50: LSTM Model (Max Performance) Confusion Matrix

3.4.5. Generated Dataset Model Variations

In addition to comparing different architectures within the CNN and LSTM models, we also conducted a comparative analysis between the two models for the jamming classification task. To evaluate the performance of both models, we specifically examined their accuracies under the different SNR values and different jamming power levels. Figures 51 and 52 show the results.

It's clear that both models aren't affected by the change in SNR or jamming power, which makes them both effective at detecting jamming early and also classifying its type, which means we can assume that the LSTM and CNN models are capable of capturing the structural characteristics of digital signals regardless of the variations in their sources (at reasonable ranges).

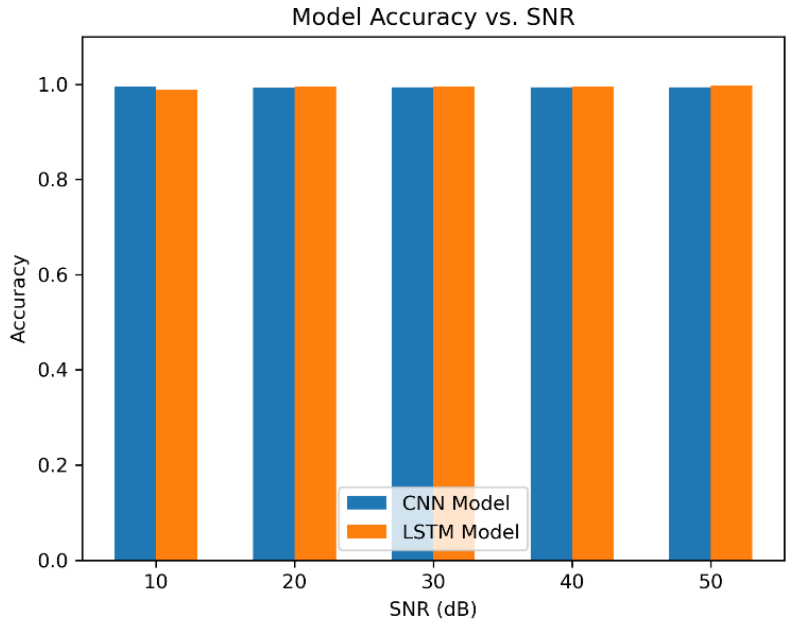

Figure 51 : Model Comparison at different SNRs on generated dataset

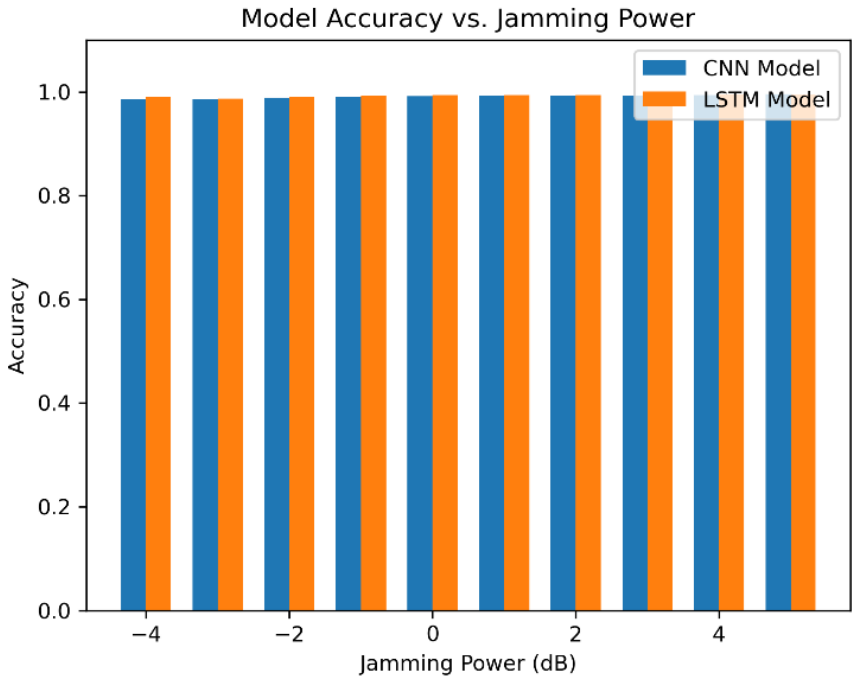

Figure 52: Model Comparison at jamming powers on generated dataset

3.5.Quantization

3.5.1. What is quantization

Quantization (figure 53) in Machine Learning is the process of converting numerical data in floating point (64 bits, 32 bits) to a fixed point with smaller precision data type like INT8 (Integer 8 bit) or INT16 and perform all critical operations like Convolution in INT8 and at the end, convert the lower precision output to higher precision in FP32 or FP64.

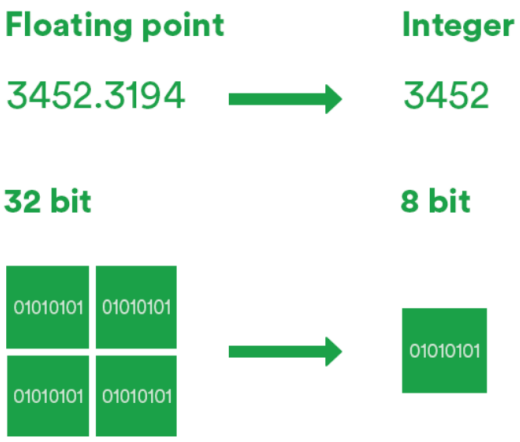

Figure 53: Quantization illustration

3.5.2. Importance of Quantization

Quantization is very important in machine learning models due to a couple of reasons:

1- Less memory and computing power needed

Quantization can dramatically reduce the memory and computation required to run the model by decreasing the precision of the model's parameters and activations. As a result, running the model on hardware with constrained resources.

2- Increased efficiency

Quantization can also enhance a model's performance by minimizing noise in the parameters and activations. Since the model is less likely to be impacted by tiny differences in the input data, this can result in more accurate and stable predictions. It can help strengthen the model's resistance to hostile cases.

3- Increased security

Quantization can partially boost security by decreasing the model's parameters' precision. An attacker may find it more challenging to obtain accurate parameter values, making it more difficult to understand how the model behaves through reverse engineering.

4- Lower power consumption

Given that power consumption is a significant concern, quantization is a crucial technique for implementing deep learning models on low-power devices like smartphones, IoT devices, and embedded systems. Quantization can also speed up inference because low-precision processes typically occur more quickly than high-precision ones.

5- Flexibility in deployment

A model becomes more versatile for deployment in many situations thanks to quantization, which also lowers a model's memory and processing requirements. Quantized models can operate effectively on various hardware architectures, including CPUs, GPUs, and specialized hardware accelerators like Tensor Processing Units (TPUs) and Application-Specific Integrated Circuits (ASICs), by decreasing the precision of the weights and activations.

6- Higher performance,

As we are dealing with 8/16 bits instead of 32/64 bits, we should be theoretically 4 times faster. In real life applications, speedup of at least 2X is observed.

Table 12: Quantization effect on speed and data size

Data Type	Accumulation	Math power	Data size reduced
Float32	Float32	1X	1X
Float16	Float32	8X	2X
INT16	INT32	8X	2X
INT8	INT32	16X	4X

Table 12 shows that the effect of quantization can speed up the math power and reduce data size greatly.

Despite the importance of quantization in machine learning models it can badly affect its performance so we should deal with this issue through careful tuning of quantization parameters as to not affect the models' performance.

3.5.3. Post-training Quantization

Quantization in machine learning can be used in two approaches: 1) quantization aware-training. 2) post-training quantization. The latter was used in our project.

After training our model, the pre-trained model is converted to a lower-precision integer representation as part of post-training quantization. The method entails evaluating a model's weights and activations on a test dataset to determine their ranges of values.

We first multiplied the models' weights and activations by a factor to save the important information as to not degrade its performance after quantization to lower precision integer.

We converted model's parameters data type to two different data types to compare between them, as shown in table 13.

Table 13: Quantization before and after data types

Before	***After***
Float 32 bits	*Integer 8 bits*
	Integer 16 bits

Illustration of the post-training quantization results for both models CNN, LSTM is given in the following context:

3.5.3.1. CNN Model

Table 14: CNN Quantization schemes comparison between INT8, INT16

Before Quantization			After Quantization			
Data type	*File size*	*Accuracy*	*Data type*	*File size*	*Accuracy*	*Math power*
Float 32	95 KB	99.46%	Integer 8	40 KB	99.2%	2X
			Integer 16	40 KB	99.4%	1X

Table [14], shows that the quantization succeeded in maintaining the good performance of the pre-trained CNN model. And memory usage is reduced to less its half.

As INT8 is better in terms of speed but with slightly lower accuracy, while INT16 can reserve the accuracy of FLOAT32 but with lower speed than INT8. This is because the quantization from FLOAT32 to INT16 and INT8 succeeded in reserving nearly all the required information for the model to perform well. Also, we can see that the file size is the same for INT8, INT16.

Figures 54 and 55 show the CNN model accuracies vs. the different SNR and jamming power values. It's shown that the quantization errors are very low, and that quantization didn't affect classification accuracies.

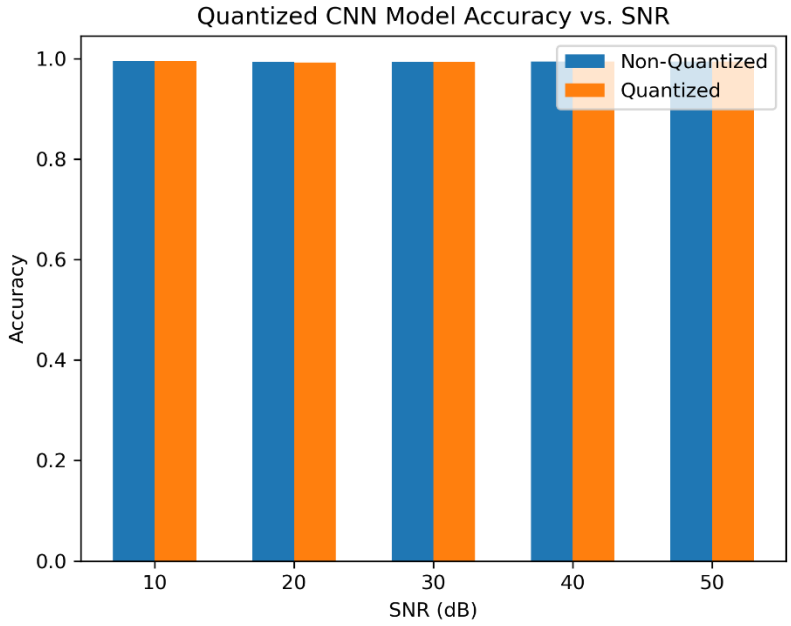

Figure 54: Quantized Model Accuracy vs. SNR

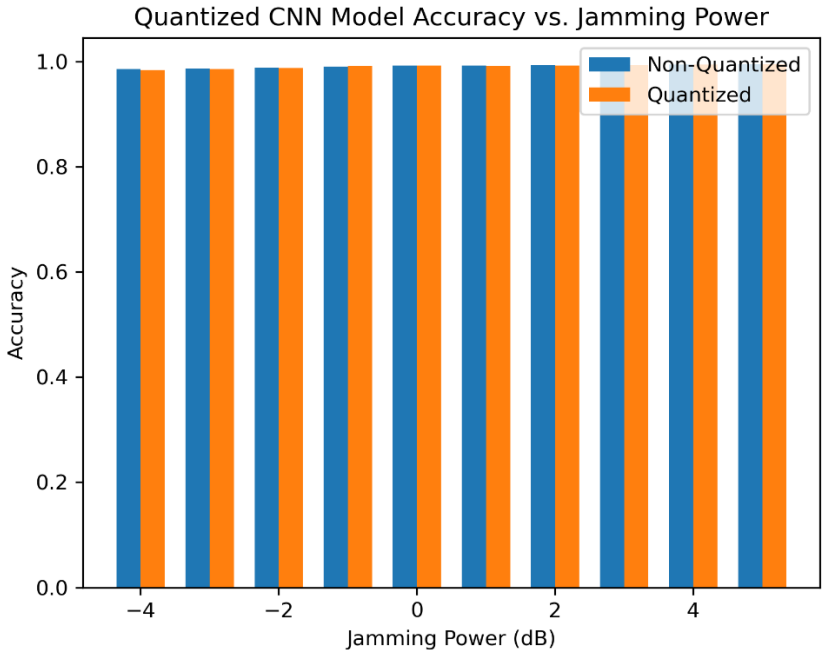

Figure 55: Quantized Model Accuracy vs. Jamming Power

3.5.3.2. LSTM Model

Table 15: LSTM Quantization schemes comparison between INT8, INT16

Before Quantization			After Quantization			
Data type	*File size*	*Accuracy*	*Data type*	*File size*	*Accuracy*	*Math power*
Float 32	41 KB	99.3%	Integer 8	20 KB	42%	2X
			Integer 16	20 KB	44.6%	1X

We can conclude from the table [15] that the post- training quantization fails to maintain the required information of weights and activations in case of LSTM model.

Note: The proposed post-training quantization technique works for the CNN model only, as the activation function used in the LSTM model (tanh) would result in performance degradation after quantization. It is worth mentioning that the LSTM model can be properly quantized but with other complex techniques.

Chapter 4 Testing on the BladeRF Board

4.1.BladeRF X40 Board Overview

As mentioned before, we trained the models on the generated data mentioned earlier. Further testing was done using the BladeRF X40 board by transmitting and receiving signals from the generated dataset to add some real-life imperfections to it and test the accuracy of jamming detection and classification after these processes and imperfections are added.

4.1.1. What is BladeRF

- BladeRF is a software-defined radio (SDR) platform designed by Naund LLC, it consists of relatively low-cost hardware components that can be programmed using software.
- It's designed to help perform good and flexible solutions for a lot of applications such as wireless communications, digital signal processing, and signal intelligence. And we're mainly concerned with Wireless communications jamming detection and classification.
- BladeRF has wide frequency range that can transmit and receive in, from 300 MHz to 3.8GHz.
- It's highly customizable and supported by many open-source software and libraries, allowing users to customize and develop their own SDR applications and algorithms.

4.1.2. BladeRF Main Components

BladeRF contains several software and hardware components such as:

- Hardware:
    1. Field-programmable gate array (FPGA): which can operate over a wide range frequency, 300 MHz to 3.8 GHz as mentioned before.
    2. Programmable RF transceiver.
    3. USB 3.0 interface: for control and data transfer.
    4. Antenna ports: for users to connect separated antenna with appropriate frequency range based on their application.
    5. Power supply port: which requires 6-18Volt DC power supply with minimum current rate of 2A

- Software:
    1. BladeRF common line interface (CLI): it is used to flash firmware files, load FPGA bitstreams, and perform other tasks on the BladeRF software SDR system
    2. BladeRF application programming interface (API): it's a software library that provides high level programming interface for controlling and interacting with BladeRF.
    3. Firmware BladeRF: it runs on the FPGA chip and provides the low-level functionality required for the operation of the hardware. The firmware can be updated using the BladeRF CLI or API.

4.1.3. BladeRF Modes

The BladeRF x40 supports several modes of operation such as follows:

1. Loopback mode: it's mainly used to send signals out in one channel and receive it back from another one. This mode is suitable for testing and calibration of the system.
2. Self-test mode: it's a built-in test that is mainly used to test the hardware and firmware functionality, it also checks various components of the BladeRF, such as ADC and DAC, RF transceiver and other components, and report any error or failure.
3. Streaming mode: this mode continuously streams data from ADC to the host pc, this is useful for applications that need high speed or real-time streaming such as real-time signal processing applications.
4. RF-no dc mode: this removes dc offset from received RF signals. This is useful for applications such as receiving FM radio signal where removing DC offset helps in reducing distortion in the demodulated signal.
5. Calibration mode: it's used to calibrate DC offset and IQ imbalance of the RF transceiver which helps in minimizing errors of the received signal.

In our case, we're interested in the loopback mood as we want to test models trained on generated data but after adding some imperfections to it.

The BladeRF X40 has several loopback modes:

1. Baseband loopback mode (figure 56): in this mode, BladeRF is configured to send a signal and receive it back through a loopback connection. This mode allows the transmitted signal to pass through DAC, TXLPF, TXVGA, RXLPF, RXVGA components but its signal doesn't pass to the next stage in the chip.

2. RF loopback mode: using this mode, BladeRF can be configured to send and receive a RF signal through TX RX channel, but with modulating RF carrier with our transmitted signal in between and demodulating it in receiving process. The signal can pass through DAC, TXLPF, TXVGA, TXMIX, AUXPA, RXMIX, RXVGA, RXLPF, RXLPF, ADC components on the BladeRF X40 chip.

3. Mix loopback mode: in this mode, the BladeRF x40 can be configured to loop the intermediate frequency (IF) signals between the TX and RX channels. This mode allows for testing the RF mixing and downconversion, upconversion functionality of the system.

4. Full Duplex Loopback: In full duplex loopback mode, the BladeRF x40 can be configured to loop the RF signals between the two channels while simultaneously transmitting and receiving data. This mode allows for testing the system's full-duplex functionality.

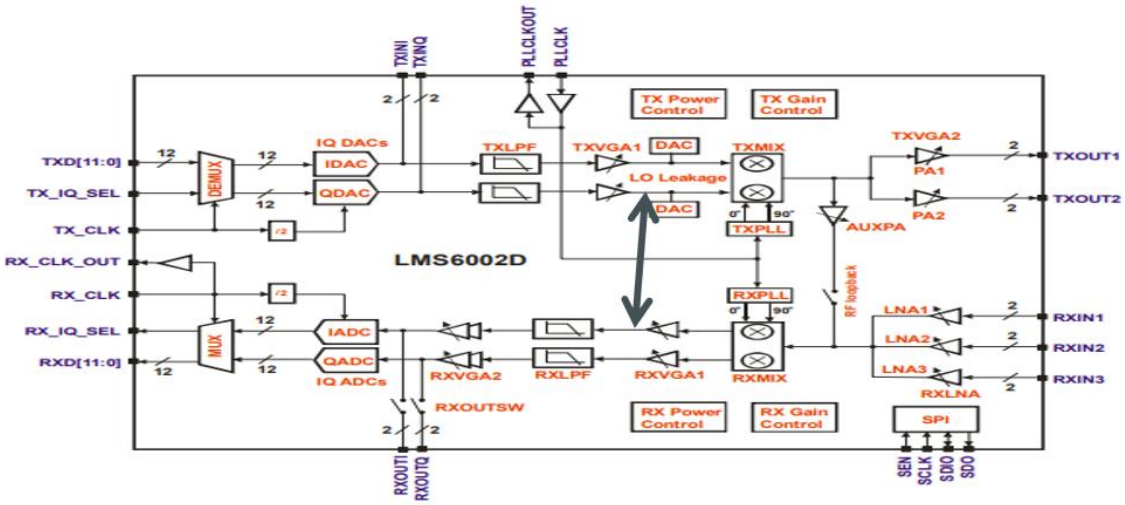

Figure 56: Baseband mode in BladeRF x40 functional block diagram

4.1.4. BladeRF Configuration Parameters

It's the BladeRF various settings that can be adjusted to configure the operation of the board. These parameters are responsible for multiple things such as choosing the mode and choosing parameters to minimize noise.

The BladeRF main parameters are:

1. BladeRF mode: to choose mode as previously explained.
2. Tx, Rx Voltage gain amplifier (VGA): the amplifier gain applied to the signal, which can be adjusted to optimize the signal strength and quality.
3. Sample rate: the rate at which the BladeRF samples the input signal, which is used to optimize the trade-off between bandwidth and resolution.
4. Bandwidth (BW): the frequency range at which the BladeRF can operate, this value is adjusted to match the input signal bandwidth.
5. Frequency: the frequency to transmit and receive at.

Before we test our models on the BladeRF-sent data, we first need to test the BladeRF itself.

So, the first step was to transmit an audio signal on the baseband loopback mode and optimize the configuration parameters to minimize noise on the received audio.

In our case, we firstly choose Baseband mode with the following parameters:

- TXVGA1=-4,
- RXVGA2=15,
- Sample rate at TX= 15M,

- Sample rate at RX= 40M,
- BW at TX= 5MHz,
- BW at RX= 20MHz,
- Frequency at TX= 2.420GHz,
- Frequency at RX= 2.412GHz.

These parameters gave us minimum apparent noise when we tested on a simple audio.

4.2. Test Results in Baseband Loopback Mode

Our goal was to transmit and receive our generated data using the BladeRF X40 board and then test our model on the received data. We transmitted all the possible scenarios presented in the generated dataset on the board:

1. No-jamming class: Tx SNR [10,20,30,40,50] dB.
2. Gaussian class: Tx SNR [10,20,30,40,50] dB with Jamming Power [-4,-3,-2,-1,0,1,2,3,4,5] dB.
3. Tone class: Tx SNR [10,20,30,40,50] with Jamming Power [-4,-3,-2,-1,0,1,2,3,4,5] dB.

Results for the 105 scenarios are tested with different SNR and jamming power values.

Using the previous configurations, we tested the received data on CNN and LSTM models. Tables 16 and 17 represent the test results for the two models.

4.2.1. BladeRF Test Results on BB mode on CNN model

Table 16: CNN model test accuracies on generated data

Base Band mode		CNN Model Test accuracy		
Tx SNR	Jamming power dB	No-jamming	Gaussian Jammed	Tone jammed
10	-4	99.80%	100%	62.85%
	-3		100%	62.85%
	-2		100%	60.80%
	-1		100%	71.74%
	0		100%	69.69%
	1		100%	67.64%
	2		100%	70.77%
	3		100%	83.28%
	4		100%	86.21%
	5		100%	79.96%

20	-4	81.66%	100%	75.75%
	-3		100%	71.55%
	-2		100%	80.35%
	-1		100%	78.10%
	0		100%	84.35%
	1		100%	81.23%
	2		100%	85.04%
	3		100%	89.05%
	4		100%	85.53%
	5		100%	89.34%
SNR	Jamming power dB	No-jamming Accuracy	Gaussian Jammed accuracy	Tone jammed accuracy
30	-4	95.87%	100%	79.86%
	-3		100%	73.21%
	-2		100%	78.00%
	-1		100%	80.44%
	0		100%	86.80%
	1		100%	83.47%
	2		100%	82.79%
	3		100%	79.96%
	4		100%	87.19%
	5		100%	85.14%
40	-4	75.62%	100%	77.02%
	-3		100%	83.87%
	-2		100%	79.37%
	-1		100%	81.91%
	0		100%	86.70%
	1		100%	81.81%
	2		100%	87.39%
	3		100%	84.35%
	4		100%	78.00%
	5		100%	86.80%
50	-4	96.21%	100%	70.28%
	-3		100%	76.73%
	-2		100%	77.90%
	-1		100%	83.87%
	0		100%	85.82%
	1		100%	78.39%
	2		100%	86.80%
	3		100%	86.90%
	4		100%	84.16%
	5		100%	89.83%

Table 16 shows us that No-jamming data accuracy gave an average accuracy of 83.8% which is good enough as BladeRF added random noise to the signals, Gaussian jammed data gave us accuracy 100% which is perfect to detect that there is jamming going to happen and its type is Gaussian. This is because the noise added by the board has gaussian distribution.

For Tone jammed data, at SNR 10, the low jamming power i.e., (-4 to 0 dB) gave a relatively low accuracy and this is mainly because the jamming power is not enough to recognize the pattern.

For all SNR higher than 10, the model started to act better for all jamming powers in tone jammed data, the overall average accuracy for tone jammed data is 82.16%, which is good enough to predict there is jamming going to happen and its type is Tone.

4.2.1. BladeRF Test Results on BB mode on LSTM Model

Table 17: LSTM model test accuracies on generated data

Base Band mode		LSTM Model Test		
Tx SNR	Jamming power dB	No-jamming	Gaussian Jammed	Tone jammed
10	-4	98.33%	100%	38.22%
	-3		100%	52.30%
	-2		100%	63.34%
	-1		100%	82.70%
	0		100%	77.32%
	1		100%	73.41%
	2		100%	72.34%
	3		100%	64.61%
	4		100%	66.47%
	5		100%	66.86%
20	-4	76.09%	100%	85.34%
	-3		100%	77.03%
	-2		100%	92.38%
	-1		100%	89.54%
	0		100%	75.76%
	1		100%	75.27%
	2		100%	73.51%
	3		100%	79.96%
	4		100%	77.71%
	5		100%	77.13%
30	-4	94.70%	100%	95.31%
	-3		100%	66.96%
	-2		100%	86.22%
	-1		100%	74.39%
	0		100%	76.54%
	1		100%	85.83%
	2		100%	84.07%

	3		100%	81.43%
	4		100%	79.47%
	5		100%	74.39%
40	-4	99.31%	100%	90.91%
	-3		100%	96.29%
	-2		100%	93.26%
	-1		100%	84.56%
	0		100%	74.78%
	1		100%	87.78%
	2		100%	81.43%
	3		100%	84.07%
	4		100%	78.98%
	5		100%	82.01%
50	-4	99.68%	100%	50.64%
	-3		100%	81.52%
	-2		100%	70.48%
	-1		100%	77.71%
	0		100%	78.01%
	1		100%	78.01%
	2		100%	87.98%
	3		100%	79.28%
	4		100%	80.25%
	5		100%	79.28%

As we can see from table 17 above, the accuracy is still accepted and good for categories No-jamming and Gaussian Jamming, but for tone jamming the average accuracy is now 77.65% which is lower than the CNN average accuracy at tone jamming data.

4.2.2. BladeRF Test Results Comparative Analysis

The following figures summarize the previous test results in the form of a comparative analysis.

4.2.2.1. Model Accuracies vs. SNR

Figures 57, 58, and 59 refer to model accuracies on the BladeRF-sent No-jamming, Gaussian jamming, and Tone jamming datasets respectively with respect to the SNR value.

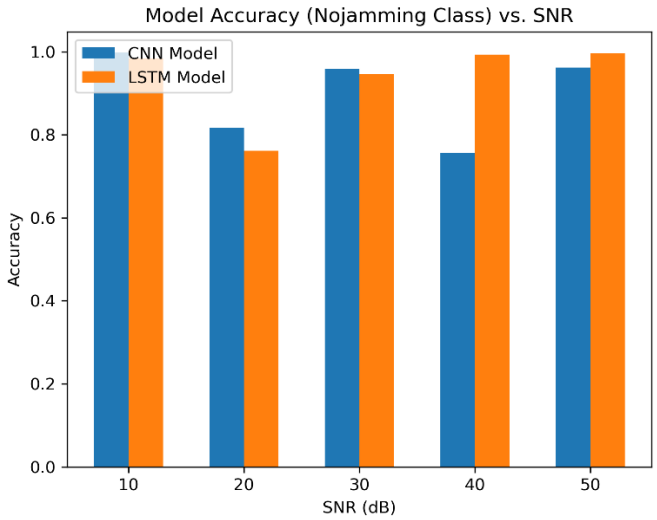

Figure 57: BladeRF-sent No-jamming Accuracies vs. SNR Comparison

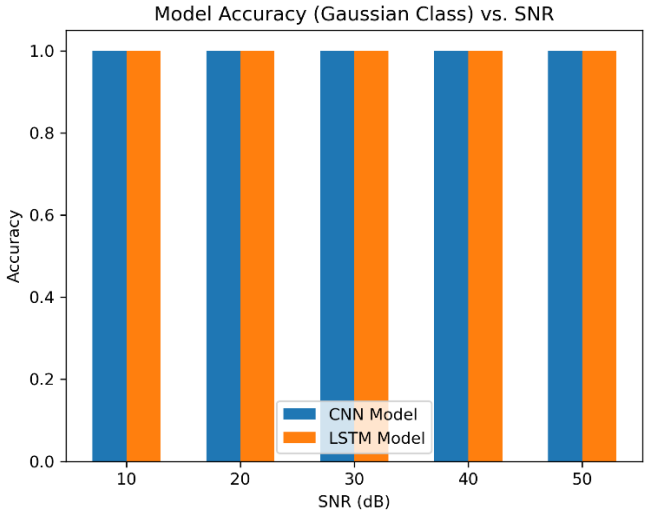

Figure 58: BladeRF-sent Gaussian jamming Accuracies vs. SNR Comparison

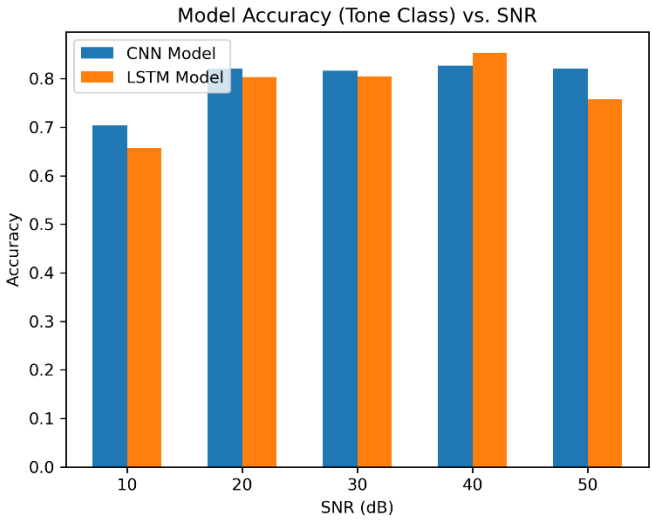

Figure 59: BladeRF-sent Tone jamming Accuracies vs. SNR Comparison

4.2.2.2. Model Accuracies vs. Jamming Power

Figures 60 and 61 refer to model accuracies on the BladeRF-sent Gaussian jamming, and Tone jamming datasets respectively with respect to the jamming power value.

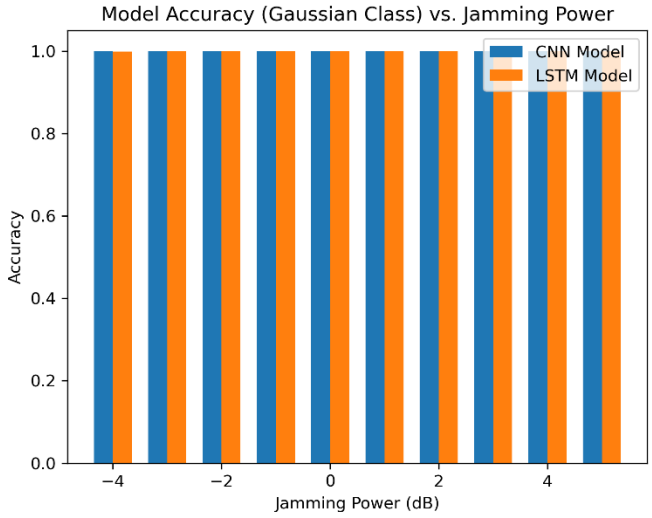

Figure 60: BladeRF-sent Gaussian jamming Accuracies vs. Jamming Power Comparison

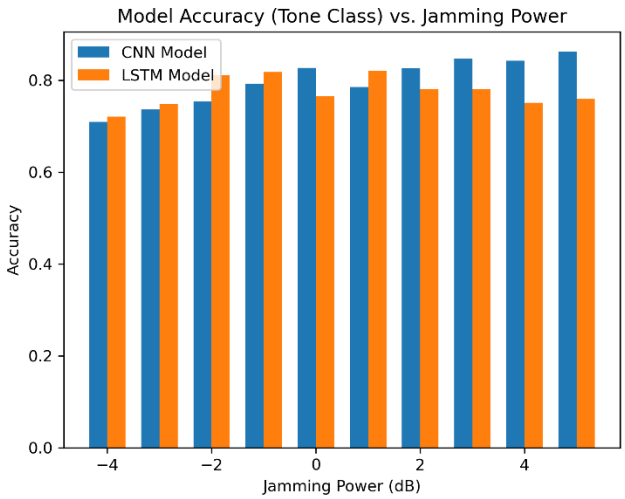

Figure 61: BladeRF-sent Tone jamming Accuracies vs. Jamming Power Comparison

4.2.3. BladeRF Test Results on CNN Quantized Model

After CNN model quantization mentioned earlier, we tested the quantization error for this model using the BladeRF-sent dataset. Table 18 shows test results for quantized CNN model.

Table 18: CNN quantized model test accuracies on generated data

Base band mode		Quantized CNN Model Test		
Tx SNR	Jamming power dB	No-jamming	Gaussian jammed	Tone jammed
10	-4	99.80%	100%	54.64%
	-3		100%	66.18%
	-2		100%	63.93%
	-1		100%	73.80%
	0		100%	71.95%
	1		100%	69.11%
	2		100%	72.92%
	3		100%	85.63%
	4		100%	88.86%
	5		100%	82.40%
20	-4	76.19%	99.90%	78.40%
	-3		100%	76.54%
	-2		100%	82.50%
	-1		100%	79.86%
	0		100%	87.00%
	1		100%	82.99%
	2		100%	87.00%
	3		100%	90.81%
	4		100%	87.59%
	5		100%	91.01%
30	-4	97.89%	100%	83.09%

	-3		100%	77.71%
	-2		100%	81.23%
	-1		100%	82.60%
	0		100%	88.37%
	1		100%	85.34%
	2		100%	83.77%
	3		100%	82.50%
	4		100%	89.35%
	5		100%	87.49%
40	-4	67.30%	99.90%	78.40%
	-3		100%	85.04%
	-2		100%	80.74%
	-1		100%	84.16%
	0		100%	88.66%
	1		100%	83.58%
	2		100%	89.05%
	3		100%	86.41%
	4		100%	79.96%
	5		100%	88.95%
50	-4	93.06%	100%	75.07%
	-3		100%	79.86%
	-2		100%	81.13%
	-1		100%	87.00%
	0		100%	88.27%
	1		100%	80.16%
	2		100%	88.76%
	3		100%	88.86%
	4		100%	87.10%
	5		100%	91.79%

We can see from the table above that after model quantization the accuracy is not highly affected in CNN model as we can see that mean accuracy for Gaussian jammed data is still 100% and Tone jammed data now have mean accuracy of 82.15% which is nearly the same as mean tone data accuracy in unquantized CNN model (82.16%), so now we have less memory usage and high inference speed after quantization with maintaining high accuracy as well.

Figures [62-66] present the test results on CNN model before and after quantization of the three classes.

- No-jamming Class After Quantization

Figure 62 shows that the model has a very close behavior before and after quantization, means that the quantization error is acceptable in No-jamming class.

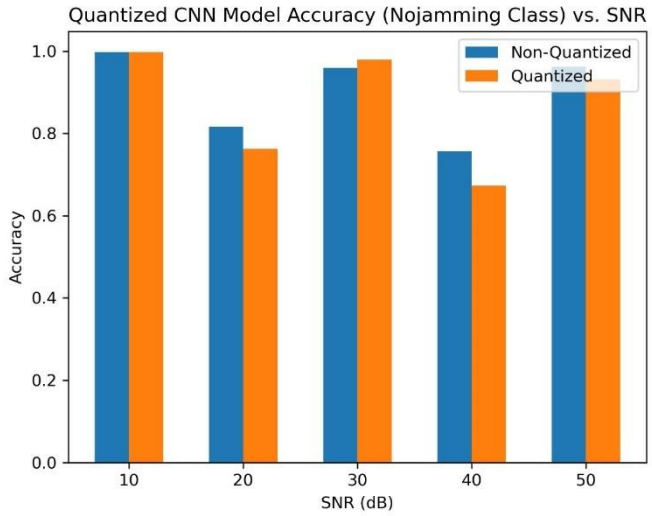

Figure 62: BladeRF Quantized CNN Model Accuracy for No jamming vs SNR

- Gaussian Jamming After Quantization

Figures 63 and 64 show that both CNN quantized and unquantized models can detect gaussian jamming with accuracy 100% approximately w.r.t Tx SNR and jamming power, means that the quantization error is acceptable in this class.

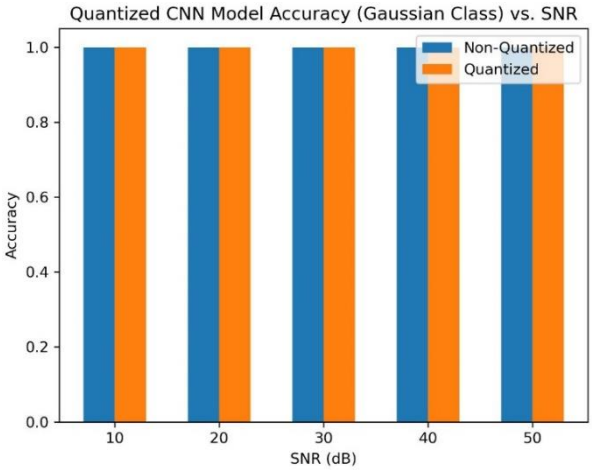

Figure 63: BladeRF Quantized CNN Model Accuracy for Gaussian jamming vs SNR

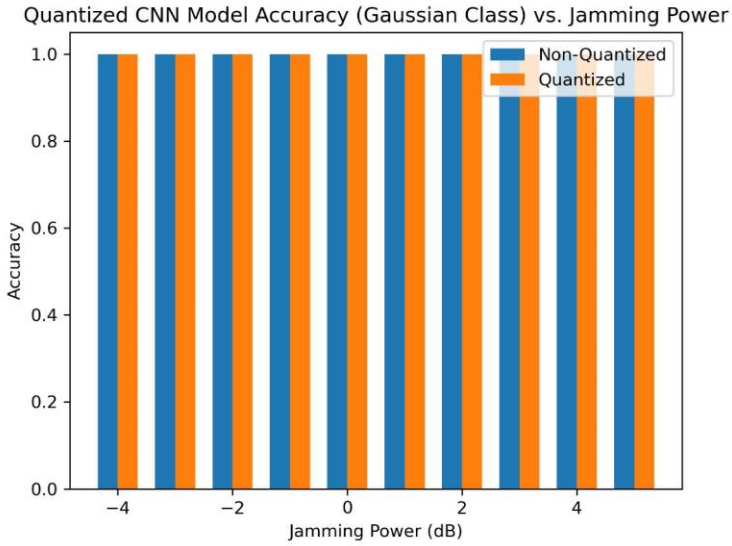

Figure 64: BladeRF Quantized CNN Model Accuracy for Gaussian jamming vs jamming power

- Tone Jamming After Quantization

Figures 65 and 66 show the effect of quantization on the CNN model in detecting tone jamming class w.r.t Tx SNR and jamming power.

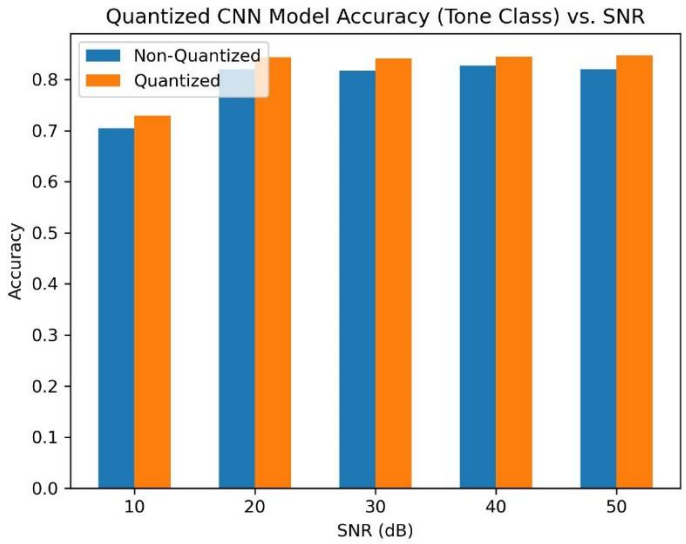

Figure 65: BladeRF Quantized CNN Model Accuracy for Tone jamming vs SNR

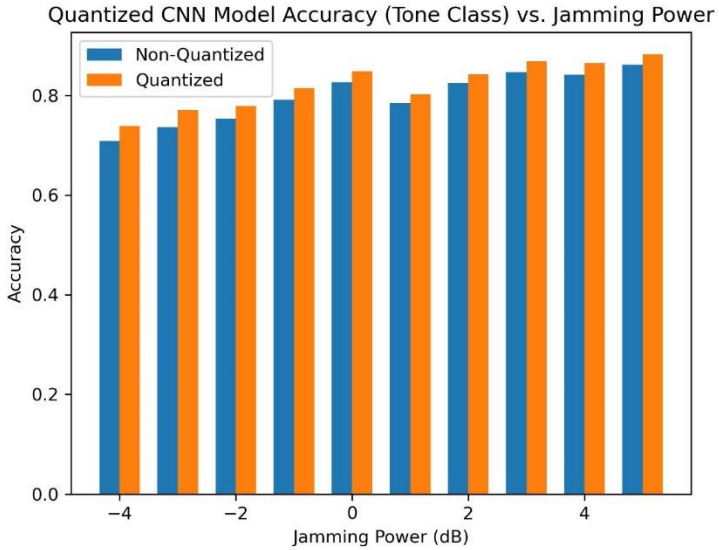

Figure 66: BladeRF Quantized CNN Model Accuracy for Tone jamming vs jamming power

We can see from figures 65 and 66, that the CNN quantized model performed better in tone jamming detection than the unquantized CNN model, this is because quantization can help to reduce the impact of numerical errors that can arise during multiplication and addition operations in deep neural networks leading to more accurate results.

4.3. BladeRF Test Conclusion

We used BladeRF x40 board to transmit and receive the generated data and then test our models as the received data from the bladeRF has some real-life imperfections and random noise coming from the board components that will be a good test for the trained models, to test whether the models are overfitted on the data or whether the generated data is too pure compared to the real-life datasets.

We got good results after testing the bladeRF in base band mode on all available scenarios in the generated dataset, for the CNN model has average test accuracies 83.8%, 100%, 82.16% for No-jamming, Gaussian jamming, Tone jamming respectively. Meaning the model is not overfitted on the trained generated data. For LSTM model it has very close performance to the CNN model in No-jamming and Gaussian classes but has lower detection accuracy in case of Tone jamming equals 77.65%.

We also tested the quantized CNN model using BladeRF board and had very similar performance of unquantized CNN model in the three classed detection, which means that the quantization error is acceptable for this model.

Conclusion

In this study, we utilized the USRP dataset [4] to train and test our deep learning models for wireless jamming detection and classification. The results demonstrated the effectiveness of the models in accurately detecting and classifying different types of jamming attacks. However, it was observed that the models trained solely on the high SNR samples from the USRP dataset struggled to differentiate between poor channel conditions and actual jamming attacks.

To address this limitation, we generated a simulated dataset that encompassed both high SNR and low SNR samples across a range of jamming powers. Training the models on this diverse dataset led to improved performance and better generalization on novel data.

Additionally, we explored the quantization of CNN model weights into lower precisions, and remarkably, the test accuracies remained high, suggesting the potential for efficient implementation in resource-constrained environments. This finding suggests that we can effectively reduce the model's memory footprint and computational requirements without significant degradation in performance. The ability to quantize the model opens up opportunities for efficient deployment in resource-constrained environments where computational resources are limited.

To validate the models in real-world scenarios, we leveraged the BladeRF X40 board and employed the baseband loopback mode to transmit the generated dataset samples. The performance evaluation on these real-world samples showed promising results, with the generated dataset-trained models and the CNN quantized model achieving relatively-high test accuracies.

These findings highlight the importance of utilizing diverse datasets that encompass a variety of jamming scenarios, allowing the models to learn robust features and generalize effectively. Moreover, the successful quantization of the CNN model weights demonstrates the potential for optimized model deployment without compromising accuracy.

In conclusion, our project demonstrates the effectiveness of deep learning models in wireless jamming detection and classification. By training on diverse datasets and exploring techniques such as quantization, we can improve the models' performance and facilitate their practical implementation.

Future Work

- Testing Models using the RF mode in the BladeRF board: as it is the nearest simulation to a real channel, as in this mode the signal is sent on a carrier signal then it will be received and converted again to the base band. So, it will be a great addition to foolproof the classification models.

- Try more types of jamming: build more general models that are capable of detecting various types, and not just gaussian and tone jamming.

- Try more modulation schemes other than BPSK.

- Experiment with OFDM symbols, as they have proved to be robust against many types of interference, so it would be interesting to see the effect of using OFDM on the application.

- Use other machine learning models, supervised and unsupervised, and do a large comparative analysis between them.

- Collect real data in various channel conditions for the purpose of training, as the generated data worked well for the purpose of our project, but there's no denying that a real training dataset would be much more robust.

- Use Data Augmentation: a technique that results in an even more diverse dataset, which can help deep learning models generalize better.

References

[1] H. Pirayesh and H. Zeng, "Jamming Attacks and Anti-Jamming Strategies in Wireless Networks: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials, vol. 24, no. 2, pp. 767-809, Secondquarter 2022, doi: 10.1109/COMST.2022.3159185.

[2] M. Cheng, Y. Ling, W. Wu, “Time series analysis for jamming attack detection in wireless networks,” IEEE global Commun. Conf., pp. 1-7, 2017.

[3] Y. Arjoune, F. Salahdine, M. S. Islam, E. Ghribi and N. Kaabouch, "A Novel Jamming Attacks Detection Approach Based on Machine Learning for Wireless Communication," 2020 International Conference on Information Networking (ICOIN), Barcelona, Spain, 2020, pp. 459-464, doi: 10.1109/ICOIN48656.2020.9016462.

[4] Alhazbi, S., Sciancalepore, S. and Oligeri, G. (2023) ‘A dataset of physical-layer measurements in indoor wireless jamming scenarios’, Data in Brief, 46, p. 108773. doi:10.1016/j.dib.2022.108773.

[5] Mathuranathan (2020) Simulate additive white gaussian noise (AWGN) channel, GaussianWaves. Available at: https://www.gaussianwaves.com/2015/06/how-to-generate-awgn-noise-in-matlaboctave-without-using-in-built-awgn-function/ (Accessed: 04 July 2023).

[6] Wickert, M.A. (2011) ‘Digital communication with jamming experiments for a signal processing first course’, 2011 Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE) [Preprint]. doi:10.1109/dsp-spe.2011.5739194.

[7] Gecgel, Selen, Caner Goztepe, and Gunes Karabulut Kurt. "Jammer detection based on artificial neural networks: A measurement study." *Proceedings of the ACM Workshop on Wireless Security and Machine Learning*. 2019.

[8] Yu, Yong, et al. "A review of recurrent neural networks: LSTM cells and network architectures." *Neural computation* 31.7 (2019): 1235-1270.

[9] Van Houdt, Greg, Carlos Mosquera, and Gonzalo Nápoles. "A review on the long short-term memory model." *Artificial Intelligence Review* 53 (2020): 5929-5955.

[10] Alhazbi, Saeif, Savio Sciancalepore, and Gabriele Oligeri. "A Dataset of physical-layer measurements in indoor wireless jamming scenarios." *Data in Brief* 46 (2023): 108773.

[11] Alzubaidi, Laith, et al. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions." Journal of big Data 8 (2021): 1-74.

[12] Brownlee, J. (2020) How to reduce overfitting with dropout regularization in Keras, MachineLearningMastery.com. Available at: https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/ (Accessed: 08 July 2023).

[13] Alhazbi, Saeif, Savio Sciancalepore, and Gabriele Oligeri. "BloodHound: Early Detection and Identification of Jamming at the PHY-layer." *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE, 2023.

[14] Alzubaidi, Laith, et al. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions." Journal of big Data 8 (2021): 1-74.

[15] Geier, J. (2013) How to: Define minimum SNR values for signal coverage by Jim Geier, How to: Define Minimum SNR Values for Signal Coverage. Available at: https://www.wireless-nets.com/resources/tutorials/define_SNR_values.html (Accessed: 13 July 2023).