Spectrum Sensing for Wireless Signal Classification and Anomaly detection

---

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

In

Electronics and Electrical Communications Engineering

Team Members:

Ahmed Mahmoud Ali Gad

Ahmed Mahmoud Abusaif

Ahmed Mokhtar Mahfouz

Gehad Mohamed Hassan

Mahmoud Taher El-Sayed

Under supervision of

Dr. Hassan Mostafa

Faculty of Engineering, Cairo University

Giza, Egypt

July 2022

This page is intentionally left blank

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BPSK	Binary Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
FC	Fully Connected
CR	Cognitive Radio
DL	Deep Learning
AM	Amplitude Modulation
FM	Frequency Modulation
PM	Phase Modulation
ASK	Amplitude Shift Keying
FSK	Frequency Shift Keying
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
PAM	Pulse Amplitude Modulation
PWM	Pulse Width Modulation
PCM	Pulse Code Modulation
SS	Signal Space
SER	Symbol Error Rate
BER	Bit Error Rate
SNR	Signal to Noise ratio
JNR	Jammer To Noise ratio
LOS	Line Of Sight
RRC	Root Raised Cosine
BW	Band Width
FFT	Fast Fourier Transform
STFT	Short Time Fourier Transform
AWGN	Additive White Gaussian Noise
ISI	Inter-symbol Interference
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
HDL	Hardware Descriptive Language
SGD	Stochastic Gradient Descent
DSP	Digital Signal Processing
SoC	System on Chip
CLB	Configurable Logic Block
LUT	Lookup Tables
MAC	Multiply Accumulate Block
MAT	Multiply Addition Block
LSB	Least Significant Bit

Acknowledgments

First and Foremost, all praise is to Allah, the Almighty, the greatest of all, on whom we ultimately depend for sustenance and guidance. We would like to thank Almighty Allah for giving us the opportunity, determination and strength to achieve this work. His continuous grace and mercy was with us throughout our life and even more during the tenure of this work.

Second, we want to thank our families for their support, tolerance, love and patience during this year especially during the hard times, they were always there for us having faith in what we do. We are grateful to our families, colleagues and friends for always motivating and encouraging us without them we wouldn't have come so far.

Third, We want to thank our major advisor Dr. Hassan Mostafa for his caring about as well as following up each stage in the project and his suggestions to solve some problems we faced during the project work. We would also like to give him special thanks for providing us to his most capabilities with any required literatures of previous or similar work in our project, hardware necessary for implementing and testing our work, and mentors to learn from and much more. Thanks a lot Dr. Hassan.

Fourth, we would like to that SeamlessWaves semiconductors for choosing us to this great project and following up with us every step on the way. Special thanks to Dr. Hassan Abushady and Dr. Tamer Khattab for their valuable ideas and advices in the project. You were great mentors and we learned a lot from you, thank you.

Finally, we would like to give special thanks to Dr. Mohammed Abdelghany, Faculty of Engineering Department of Electronics and Electrical Communications in Cairo University, for mentoring us in our early work in the project. His advices for us were great and helped us a lot, and formed the backbone of our work in the first steps of the project, thanks a lot.

Abstract

Cognitive Radio and Spectrum sensing are becoming more demanding specially after 5G technology has been released and the more it spreads the more important it will be to implement them in our systems. Our project's main task is to detect and classify anomalies in the spectrum specifically Jamming.

We created several datasets with different specifications, as there were none found online. The data set we generated had 8 jamming types; tone jamming, multi tone jamming, sweep jamming, parabolic sweep jamming, C&I jamming, ON-OFF jamming, hop jamming, barrage jamming, added to the signal. We also varied the jamming power in the data set to try to mimic real world scenarios. We then developed a Deep learning algorithm that can detect and classify the jamming in spectrum, we also built a model that can detect Modulation schemes of the signal. In our deep learning jamming detection model, we used STFT as a pre-processing function to obtain time-frequency information of the signal and used a three CNN layers to do feature extraction of these time-frequency data, and then passed them to three FC layers to do the classification. We were able to obtain an average accuracy of 56.89% over SNR levels [-20dB : 18dB] and 93.31% over SNR levels [0dB : 18dB] on the final dataset. Our next goal was to implement this deep learning model on FPGA, we first implemented the STFT by utilizing the FFT ip that Vivado provides then we went on and designed the CNN layers using different implementation for each layer to maximize performance and obtain high accuracy and fast operation.

Chapter 1: Introduction

1.1 Cognitive Radio and Spectrum Sensing

In today's world, the use of wireless devices has increased significantly with the advances in wireless technology. In the near future, significant growth of connected devices is expected with the mass adoption of IoT. A huge amount of spectrum is required to support this increasing number of wireless devices. But the spectrum available is a scarce resource. If we check the current spectrum allocation chart, it's very hard to find a free spectrum to support upcoming volumes of wireless devices and mobile data traffic. Cognitive Radio is a concept introduced to tackle the upcoming spectrum crunch issue. Cognitive radio (CR) is a form of wireless communication in which a transceiver can intelligently detect which communication channels are in use and which ones are not, it also minimizes interference to other users by avoiding occupied channels, increases spectrum efficiency, and improves the quality of service for users.

The technology resulting from the merging of 5G and Cognitive Radio (CR) is effective to meet the heavy mobile data traffic of future wireless networks. The new era of communication will be dominated by 5G in the future. As the future mobile broadband will be largely driven by ultra-high definition videos and as the things around us will be always connected, 5G aims to provide higher capacity and a network speed of 10Gbps. 5G equipment will also be available at lower cost, lower battery consumption, and lower latency than 4G equipment. 5G platform can empower the growth of many industries ranging from entertainment, agriculture, IT, and manufacturing industries. The need for more capacity will demand more spectrums resulting in the integration of CR in 5G networks. The focus of CR is to enable much more efficient use of the spectrum though it adapts itself to provide the optimum communications channel.

All radio communication signals (radio, television, mobile phones, etc.) are modulated before transmission. Modulation recognition/classification is fundamental for correct demodulation. This has many applications in military, intelligence, and civil communities In recent years much emphasis are put on the development of signal processing and artificial intelligence techniques to address the relevant issues. Modulation recognition is an essential and challenging topic in the development of the cognitive radio and it is an essential part of CR adaptive modulation and demodulation capabilities to sense and learn environments and make corresponding adjustments, which make modulation recognition one of the main issues in the project. The cognitive radio (CR) system has been adopted for efficient utilization of the radio frequency spectrum. The classification of signal modulation schemes is one of the main characteristics of the CR for the appropriate demodulation of sensed signals. However, conventional Modulation Classification (MC) techniques require extensive extraction of signal features, which is not often guaranteed. Thus, Deep Learning (DL) has been seen as a promising solution to this drawback in MC. DL has shown overwhelming advantages in computer vision, robotics, and voice recognition. Recently, DL has been proposed to apply to wireless communications for signal detection and classification in order to better learn the active users for electromagnetic spectrum sharing purposes.

So, we apply in our project a convolutional neural network (CNN) which is a DL algorithm, to demonstrate the feasibility of using CNN to recognize and classify over-the-air wireless signals.

1.2 Importance of Anomaly detection

As a result of cognitive radio capabilities, one of its major applications is detecting jamming. Detecting Anomalous signals like jamming attacks is important because it is the first step toward building a secure and dependable wireless network. 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 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 interrupt of the communication due to failures at the sender side; and other similar conditions. Communication in the wireless sensor networks could be disturbed by the jammer and this jamming attack could be identified as a distinctive type of denial of Service attack. This might result in the degradation of the network which could be recognized. Due to the jamming signals, the packet transmission might not be proper. These kinds of attacks are destructive in low power due to the attributes such as disruption in communication and rapid trench in the batteries. To detect the attack and to perform secure data transmission, the illegitimate nodes have to be eradicated. Machine learning and deep learning methods are used for network analysis of intrusion detection and security. Hence, the deep learning model is proposed to identify the attacks which result in secured data transmission. Deep learning is applied so it can adaptively learn the attacks and classify them with higher accuracy. It is efficient as it is adaptive in learning and has improved precision.

1.3 Problem Statement

As discussed in the previous section the presence of anomaly in the communication link disrupts the communication and might cause all sorts of failures such as data loss or false data reception. This anomaly can be intentional in that there is an attacker who wants to jam the signals in the spectrum to disrupt the communication link so called jamming, or it can be non-intentional due to node failures in the wireless communication link. We decided to focus on jamming anomaly. Therefore, our main target is to design a deep learning algorithm and train it, using dataset of data transmitted in the channel that has been subjected to jamming, in order to be able to detect these jamming signals, we are then required to deploy this algorithm on FPGA, in order to be able in the future to implement this DL algorithm on low power and battery running devices with minimum power consumption and maximum accuracy possible, for DL algorithms require a large computational power, so it is important that when the model is implemented on FPGA it does not fail.

We decided to make not only jamming detection system but also jamming classification as well, because if the jamming type is known, it might be easier to remove its effect on the data.

1.4 Previous Work in this area

We searched for previous work in the area of using DL in jamming detection and found one that was using CNN with spectrogram images of the spectrum to classify anomaly, they were able to classify 8 types of anomalies both jamming and node failures. Their approach was to compare two images one predicted by the CNN model for the next frame based on the previous ones and the other is the real frame that has been received. If they were identical then there is no anomaly if not then there is an anomaly. The main difference between them and our work is we are aiming for a hardware implementation of the system. The other

published literature in this area were all dealing with the problem of modulation detection/classification.

1.5 Main obstacle found

As mentioned before all DL techniques require a large data set to train the model but in our project there was no data set available with jamming, we found a data set provided by deepsig online but it had no jamming. We reached out to other researchers to ask them for their dataset but there were no reply, so our only option was to generate the data set ourselves.

1.6 Chapters Overview

Chapter one gives a quick introduction about Cognitive radio, spectrum sensing, anomaly detection and discusses the problem statement if 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 specifications required for the dataset and ending with the generated datasets and their main differences.

Chapter three deals with the Deep Learning model we made starting by explaining what is deep learning and its fundamentals it also discusses the CNN with all its nuances, after that it gives a detailed illustrations of our model and ends with how we made the model more suitable for hardware implementations

Chapter four then continues with this hardware implementations and discusses methods to implement the model and enhance its accuracy FPGA then it explains what is FPGA and its main capabilities and components, it then shows our implementation of the model and its different parts

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

Chapter 2: Communication and dataset

2.1 Communication Fundamental Concepts

2.1.1 Communication System

Any communication system has three main components that is the transmitter, the channel, and the receiver as shown in figure 2-1.

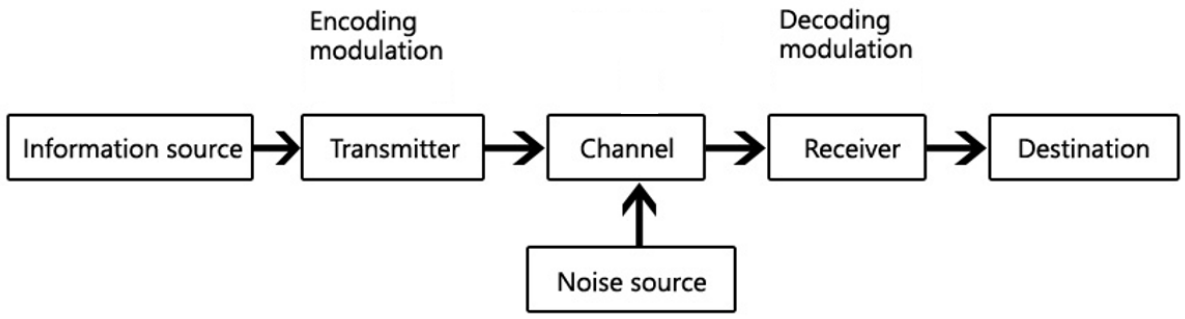


Figure 2- 1 Communication System

The information may undergo several operations in the transmitter before it is transmitted such like encoding, modulation, up sampling… etc. and then the information is transmitted into the channel. There is two types of the channel wired channel, in which the transmitted data reaches the receiver through a guided medium like copper wires, optical fiber cables… etc. and the other type is wireless channel, in which the data reaches the receiver through an unguided medium such as air. In both cases, the channel adds some noise to the data although its effect on the wired and wireless communication varies a lot. Then the data reaches the receiver whose function is to try to cancel out the changes that happened to the signal like the noise from the channel and the operations the data went through in the transmitter, so it tries to cancel the noise, down sample the signal, demodulate it, decode it … etc. to retrieve the original information and deliver it to the destination.

2.1.1 Modulation and demodulation

Modulation is the process of encoding information in a transmitted signal in the transmitter side, while demodulation is the process of extracting information from the received signal at the receiver side. Many factors influence how accurately the extracted information replicates the original input information. Electromagnetic interference can degrade signals and make the original signal impossible to extract. Demodulators typically include multiple stages of amplification and filtering in order to eliminate interference.

2.1.2 Why do we use modulation?

Modulation is used because it provides very important features as follows. First, it provides ease of transmission of the signals; because the antenna's length is inversely proportional to the transmitted signal frequency hence a smaller antenna for high frequency signal, which is easier in manufacturing and implementation in small devices like phones. Second, it allows for the ability of simultaneous transmission of multiple signals or multiplexing like for example FDM, TDM…etc., which is helpful in transmitting multiple

signals at the same time in the same geographical area without interference like TV channels, which allows for an efficiently use of available bandwidth.

2.1.3 How modulation works

Modulating a signal is mainly multiplying the signal by another high frequency carrier signal after doing some encoding on the signal to raise the signal's frequency to be the same as the carrier signal multiplied. Encoding means that information from the signals can be added to some carrier by varying its amplitude, frequency or phase. Modulation is usually applied to electromagnetic signals such as radio waves, lasers/optics and computer networks. Modulation can even be applied to a direct current, which can be treated as a degenerate carrier wave with a fixed amplitude and frequency of 0 Hz mainly by turning it on and off, as in Morse code telegraphy or a digital current loop interface.

2.1.4 Types of modulation

There are many types of modulation schemes that are used in communication systems; figure 2-2 represents summaries of some of them. We are mainly concerned in this literature with the digital signals and the modulations that can be used with it, so our concern is the analog carrier modulation digital data branch in figure 2-2.

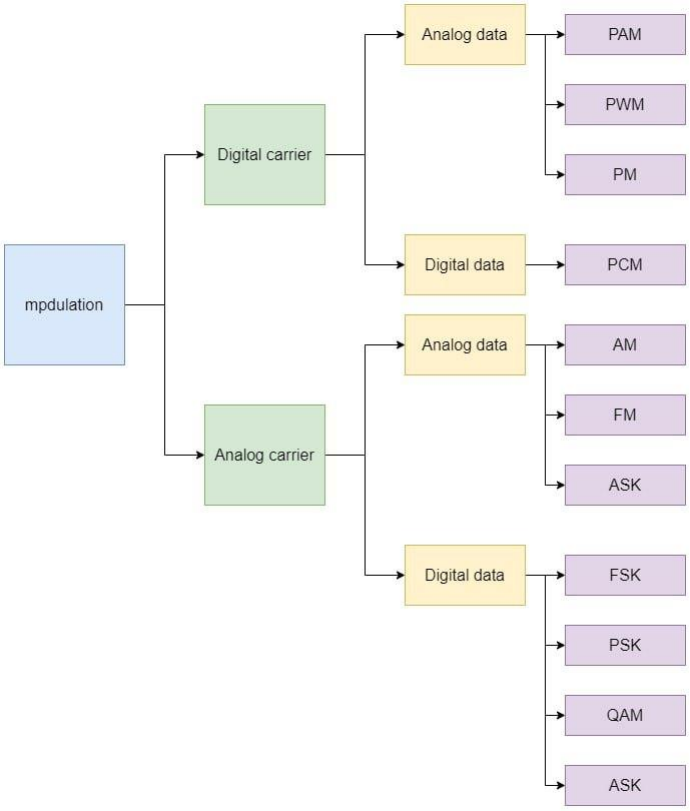


Figure 2- 2 Modulation types

2.1.5 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 for a given target BER.

2.1.6 Signal Space

Signal space is an important concept in digital modulation. The signal space of a modulation type, like shown in figure 2-3 for example, is a figure representation of all the symbols that can be generated from the modulation type specified where x-axis is the I-Phase (In-Phase component) of the data and the y-axis is the Q-Phase (Quadrature component) of the data. It is used to show the distance between symbols to judge the immunity of the modulation type used to noise and jamming. The number of symbols represented is based on modulation order and each symbol is an encoded number of bits which is mostly gray encoding; to decrease the bit error rate (BER) for the same symbol error rare (SER) and also decrease the total energy required to transmit the signal.

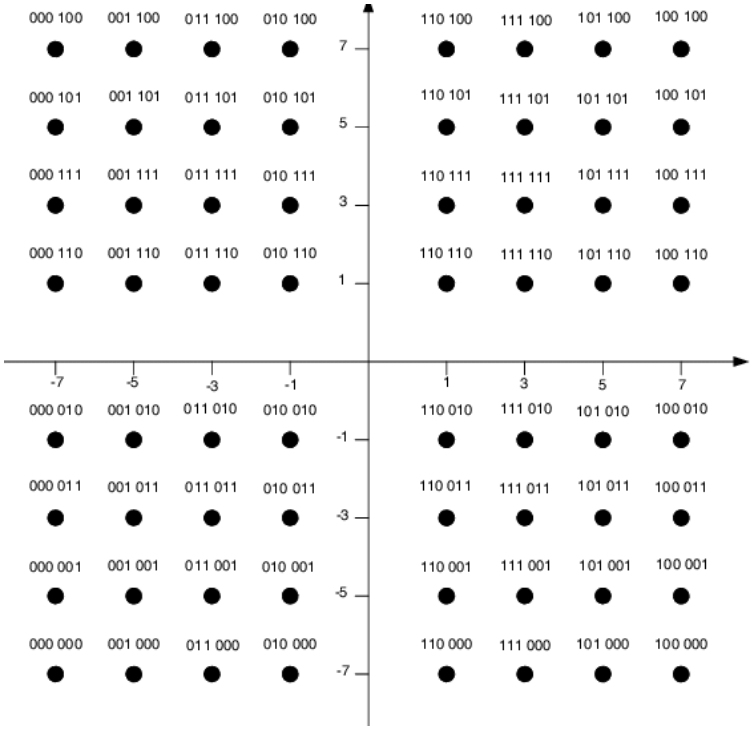


Figure 2- 3 Signal Space representation of 64QAM modulation

2.1.7 Upsampling

Upsampling is the process of inserting zero-valued samples between the original samples of the signal to increase the sampling rate. One way to accomplish upsampling by an integer ratio of 1:D is to interpose D-1 zero samples between each pair of the input samples of the signal. This causes the spectrum of the original signal to repeat at multiples of the original sampling rate. The process of upsampling doesn't change the content of the input signal, it only introduces a scaling of the time axis by a factor D. Consequently, the operation of upsampling is invertible, which means that it is possible to recover the input signal from samples of the output exactly.

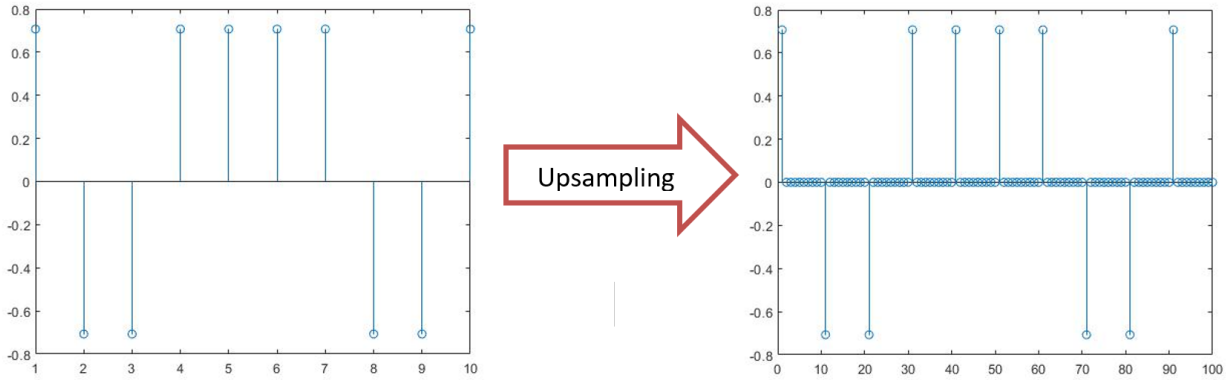


Figure 2- 4 Upsampling by a factor of 10 meaning we added 10 zeroes after each sample (note the time axis scale)

2.1.8 Pulse Shaping

In electronics and telecommunications, pulse shaping is the process of changing the waveform of transmitted pulses. Its purpose is to make the transmitted signal better suited to its purpose or the communication channel, typically by limiting the effective bandwidth of the transmission. By filtering the transmitted pulses this way, the inter-symbol interference caused by the channel can be kept in control. In RF communication, pulse shaping is essential for making the signal fit in its frequency band. Typically, pulse shaping occurs after line coding and modulation.

2.1.8.1 Pulse shaping filters

Not every filter can be used as a pulse shaping filter. The filter itself must not introduce inter-symbol interference (ISI); it needs to satisfy certain criteria. The Nyquist ISI criterion is a commonly used criterion for evaluation, because it relates the frequency spectrum of the transmitter signal to ISI.

Examples of pulse shaping filters that are commonly found in communication systems are:

1- Raised-cosine filter
2- Sinc shaped filter
3- Gaussian filter

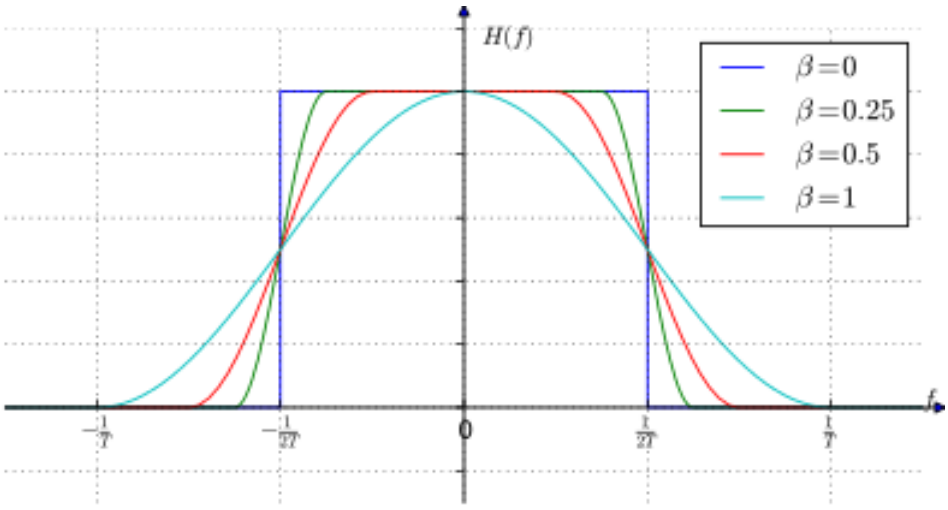


Figure 2- 5 Amplitude response of raised-cosine filter with various roll-off factors

Sender side pulse shaping is often combined with a receiver side matched filter to achieve optimum tolerance for noise in the system. In this case, the pulse shaping is equally distributed between the sender and receiver filters.

2.1.9 Fading channel models

There are two famous channels know and used in communication systems simulations:

1- Rician fading is a stochastic model for radio propagation randomness caused by partial cancellation of a radio signal by itself the signal arrives at the receiver by several different paths hence exhibiting multipath interference, and at least one of the paths is changing. Rician fading occurs when one of the paths, typically a line of sight signal or some strong reflection signals, is much stronger than the others. In Rician fading, the amplitude gain is characterized by a Rician distribution.

2- Rayleigh fading is sometimes considered to be a special case of Rician fading for when there is no line of sight signal. In such a case, the Rician distribution, which describes the amplitude gain in Rician fading, reduces to a Rayleigh distribution. Rician fading itself is a special case of two-wave with diffuse power (TWDP) fading.

In the characterization of the channel model there are two important parameter that needs to be specified:

- The ratio between the line of sight path power and the other paths power (K) as shown in figure
- The total power from all paths

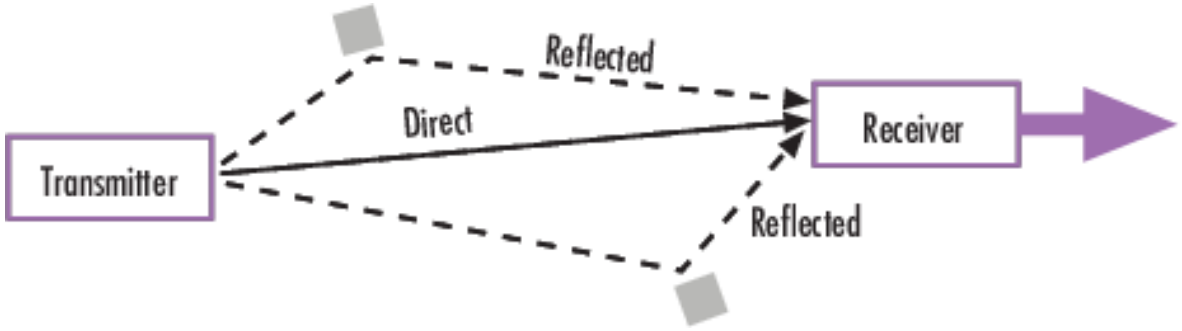


Figure 2- 6 fading physical model

2.1.10 Jamming definition

As mentioned in the problem statement, our focus is jamming detection hence it is a necessity to be aware of the definition of jamming. Jamming in wireless networks is defined as the intentional disruption of existing wireless communication by decreasing the signal-to-noise ratio at the receiver side through the transmission of wireless interfering signals. There are many types of these wireless interfering signals such as, tone jamming, multi-tone jamming, sweep jamming, barrage jamming … etc. we will illustrate them and others in an upcoming section as it is more suitable they are mentioned there not here.

2.2 Dataset Specifications and Generation

As mentioned in the previous chapter, the previous work was mainly focused on modulation detection and they had a very good dataset available online, which is RADIOML2016.10A, the only problem in that data set is that it does not have jamming in it which is a pitfall for us as we need to build a deep learning algorithm for jamming detection hence we searched for a published data set that has jamming but couldn't find any. Therefor our only remaining option was that we generate our own data set.

2.2.1 Dataset used in previous work

The data set used in the previous work on the modulation recognition is the RADIOML2016.10A which has 11 modulation schemes and consists of 220,000 In-phase and quadrature (I/Q) represented data vectors divided into 20 different SNR levels limited between [-20 : 18] dB and each frame consists of 256 sample (2 x 128).

2.2.2 Dataset used in our work

We generated our own dataset using Matlab and we were only concerned with digital modulation techniques so we generated 5 of the digital modulation techniques used in RADIOML dataset and we added another 4 digital modulations and we inherited some of the features of the RADIOML dataset such as channel type and communication link.
Figure 2-7 shows the nine digital modulation schemes included in our data set.

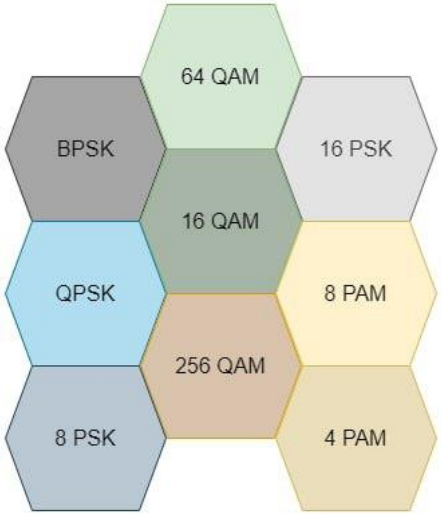


Figure 2- 7 generated digital modulation techniques

2.2.3 Data Key features changes

In order to hit higher accuracy then the previous work in the part of modulation detection there were some key components in the dataset that needed to be addressed and adjusted:

- The dataset size should be increased to ease the building and tuning of the deep learning model so for each class we increased the number of examples from 20000 in RadioML.10A to 40000 in our generated dataset.
- Also the frame size should be increased to allow the model to see and fetch more details of the data so we increased the frame size from 128 to 512 in our dataset.
- And lastly in order for the deep learning model to achieve high classification accuracy for the modulation detection it needs to see number of symbols comparable to the maximum modulation order used in the dataset which is not the case in the RadioML dataset since they used oversampling rate of around 10 for a frame of 128 sample which is around 12 symbols only and that explains why in their work the PSI model was not able to achieve accuracy higher than 85% even for the higher SNR levels.

2.2.4 Changes we did in the distribution of the dataset

The data set we generated in this literature for the purpose of modulation detection includes 9 digital modulation schemes, and consists of 360,000 In-phase and quadrature (I/Q) represented data vectors divided into 20 different SNR levels limited between [-20:18] dB and each frame consists of 1024 sample (2 x 512), so in conclusion the data is (9 mods×20 SNR level×2000 frame ×2×512).

Another important point to note is that the data is random in every dimension so that the data transmitted and the noise added is changed in every modulation type, SNR level and frame, also, the start of the frame is changed every frame to be able to mimic the real world situation fully without overfitting the model to certain patterns.

2.2.5 The Communication system built

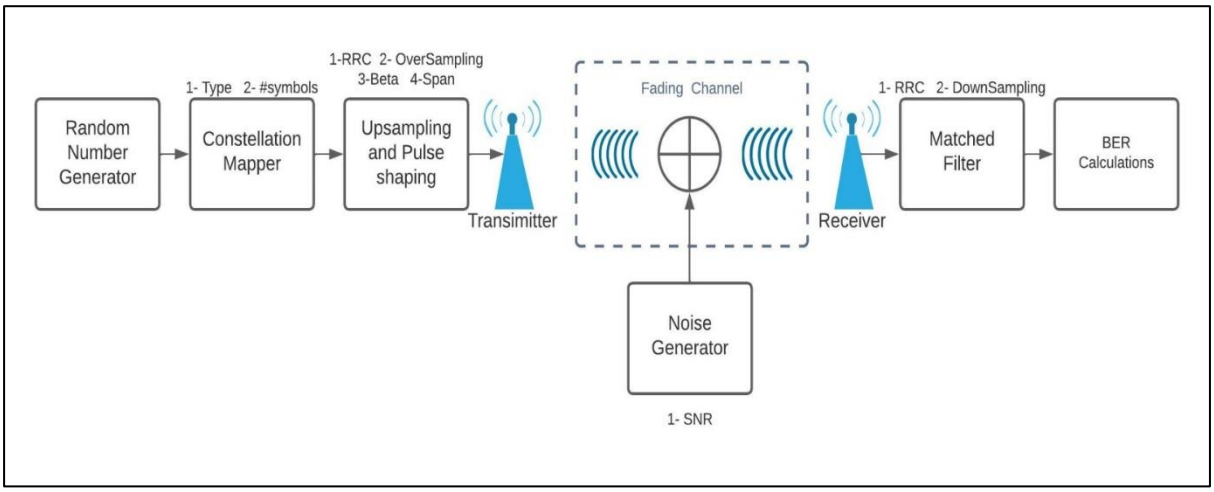


Figure 2- 8 the communication system used to generate the dataset

For the purpose of generation the data we built the communication system shown in figure 2-8 and extracted the data after the channel so that it can be used when the detection system have no idea about the pulse shaping and the other parameters of the transmission,

which is the case we are interested in, also we have generated the data after the matched filter which can be used only if the detection system is in the receiver or have knowledge about the transmission parameters, also it is interesting to see the effect of the jamming on the symbols in the signal space (SS), which gives an idea about the jamming effect on BER.

It is important to note that the power of the jamming and the noise are both calculated in the channel not after receiving of the data.

The communication system block diagram in figure 2-8 consists of the following blocks:

1- Block one
    a. Random number generator: generates the data bit stream to be transmitted.
2- Block two
    a. Constellation mapper: maps the data to symbols according to the gray encoding
3- Block three
    a. Up-sampling: (also called interpolation) increases the sampling rate ,improves anti-aliasing filter performance and reduces noise, and here we used an oversampling value of 4
    b. Pulse shaping: for the pulse shaping we used a root raised cosine (RRC) with fixed value of a roll-off factor equal to 0.5 and system span equal to 7
4- Block four
    a. Channel: which we modelled to be Rician fading channel like the RadioML dataset also, noise is added in the channel

    The channel parameters that are fixed for the generation
    - Sampling rate = 200 MHz
    - Maximum Doppler shift of diffuse components = 40 KHz
    - Discrete delays of 5-path channel = [0 4.5 8.5 12 25] (ns)
    - Average path gains = [0 -2 -10 -13 -16] (dB)
    - K Factor = 4 (Linear ratio of specular power to diffuse power)
    - Doppler shift of specular component = 20 KHz
5- Block five
    a. Matched filter: receives the data and convolves it with the pulse shaping filter again to have the response of raised cosine
    b. down-sample: transform the data back to the signal space (I/Q form) (also called decimation which helps reducing the sampling rate)
6- Block six
    a. BER calculation: compare the transmitted bits with the received bits to calculate the bit error rate (BER)

Pulse Shaping used in generation

- System span = 7
- roll of factor = 0.5

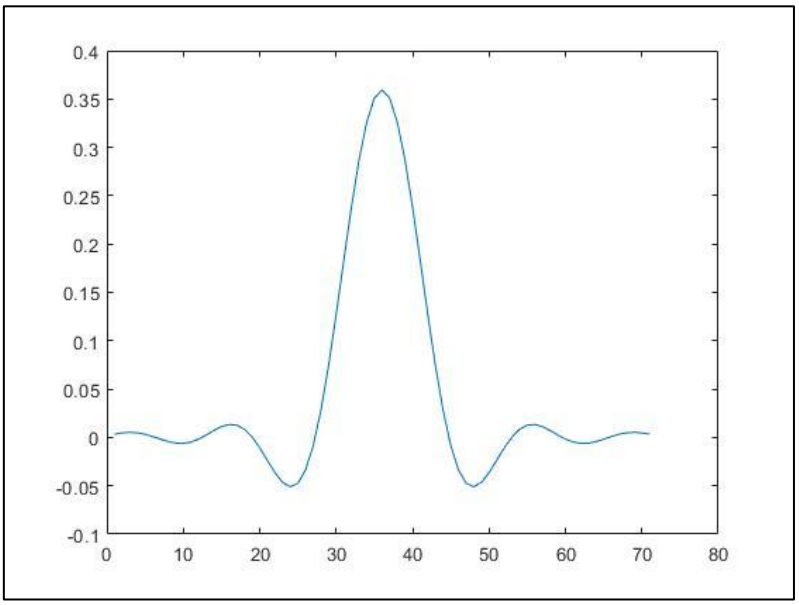


Figure 2- 9 root raise cosine shape

2.2.6 Modulation types

And the following is the signal space for each modulation normalized for average power = 1

Table 2- 1 Modulation techniques generated

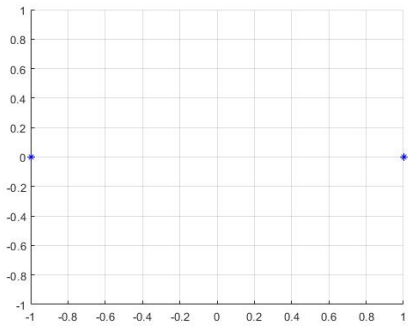
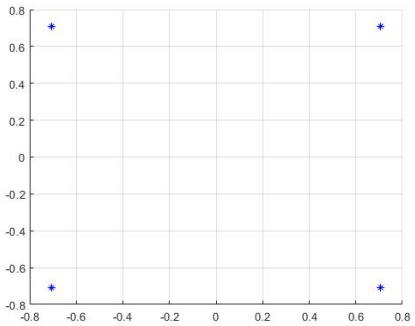
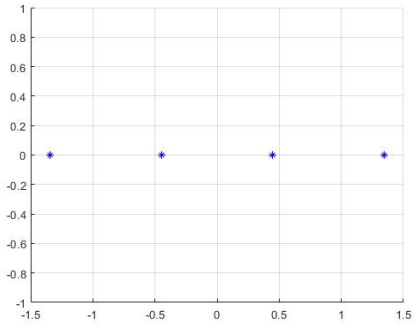
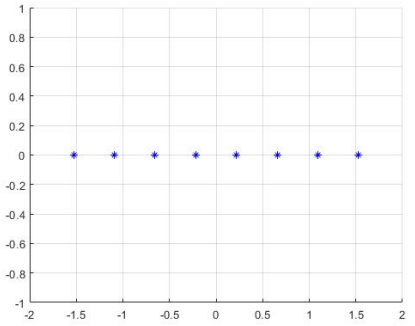
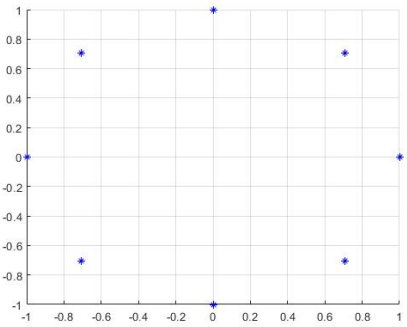
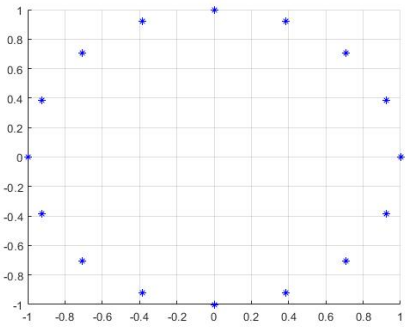
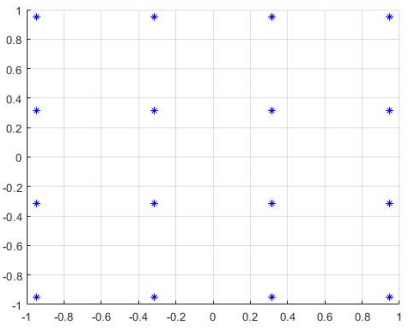
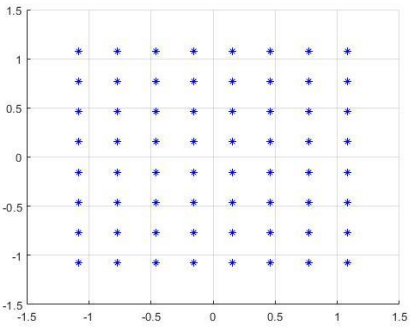
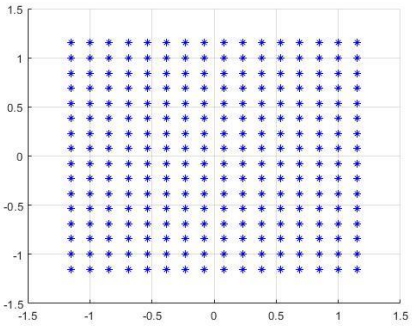
BPSK signal space	**QPSK signal space**
	
4PAM signal space	**8PAM signal space**
	

Table 2- 2 Modulation techniques generated (continued)

8PSK signal space	16PSK signal space
	
16QAM signal space	**64QAM signal space**
	

256 QAM signal space



2.2.7 BER vs SNR for the generated dataset

This is the BER vs SNR for the five digital modulation schemes used in the RadioML dataset and in our dataset

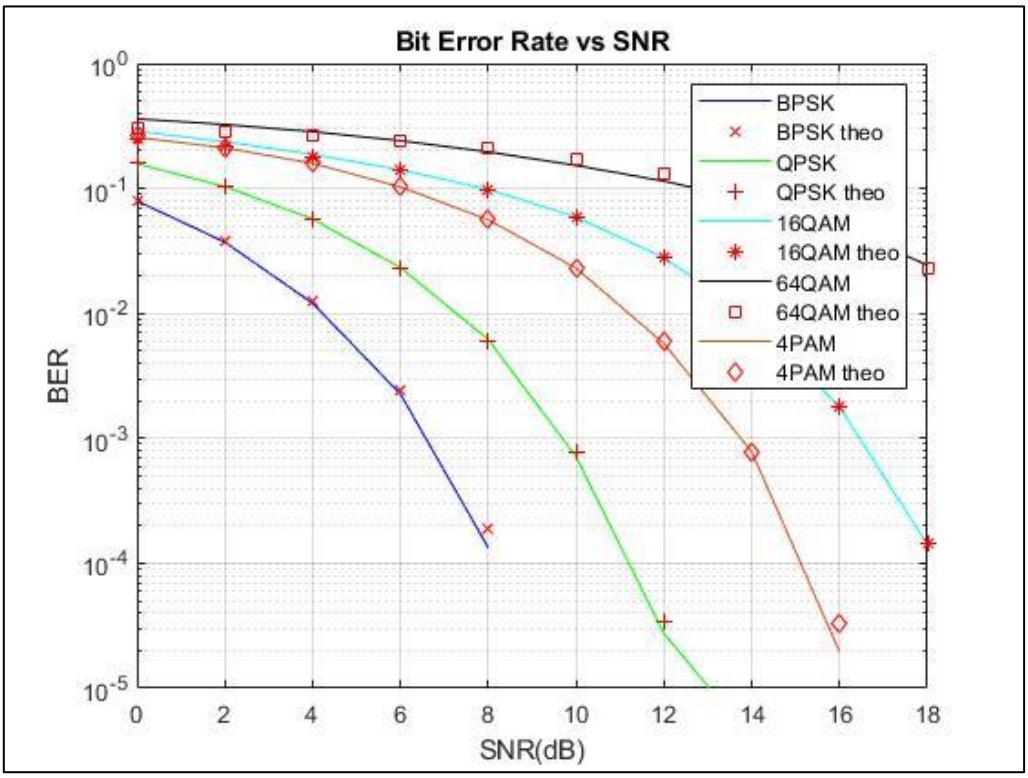


Figure 2- 10 BER of some of the modulation techniques generated compared to its theoretical value

2.2.8 Communication system plus jamming

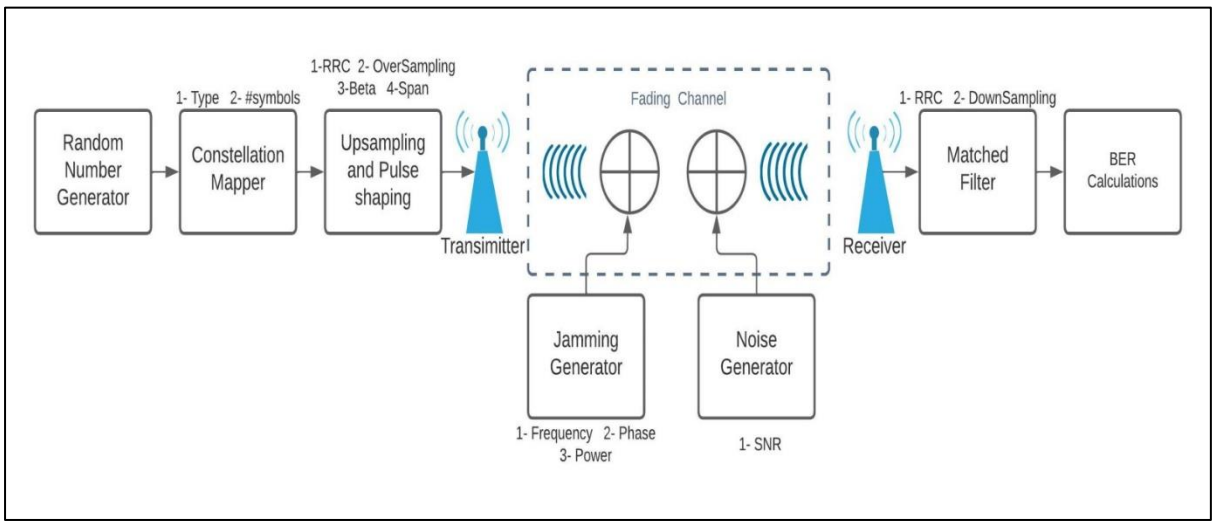


Figure 2- 11 Communication system block diagram after considering jamming effect

Since there is no published dataset for the jamming detection problem we are facing, we generated our own data to represent the problem and we inherited some of the parameters of the communication system from the RADIOML dataset

For our dataset we are concerned only with digital modulations, we generated 9 types of modulations and each one is generated 9 times one for each class of the jamming (8 jamming types + 1 normal class) which will be discussed later in this literature.

The block diagram in figure 2-11 represents the communication network we have built and used to generate the data, and shows the key parameters in the dataset.

1- Block one
    a. Random number generator: generates the bit stream data to be transmitted
2- Block two
    a. Constellation mapper: maps the data to symbols according to the gray encoding
3- Block three
    a. Up-sampling: (also called interpolation) increases the sampling rate, and improves anti-aliasing filter performance and reduces noise, and here we used oversampling value of 4
    b. Pulse shaping: for the pulse shaping we used a root raised cosine (RRC) with fixed value of a roll-off factor equal to 0.5 and system span equal to 7
4- Block four
    a. Channel: which we modelled to be Rician fading channel like the RADIOML dataset for the data signal and slow fading and frequency non-selective for the jamming signal, and also, noise is added in the channel.

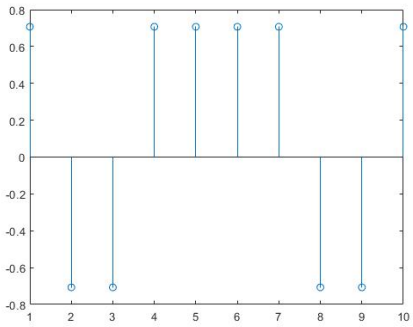
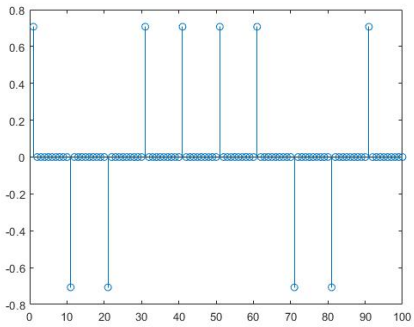
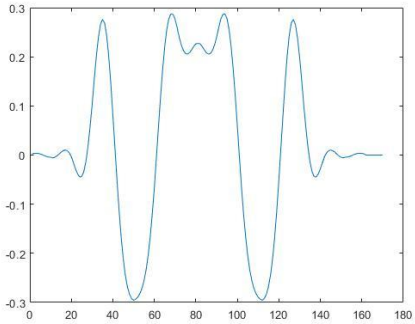
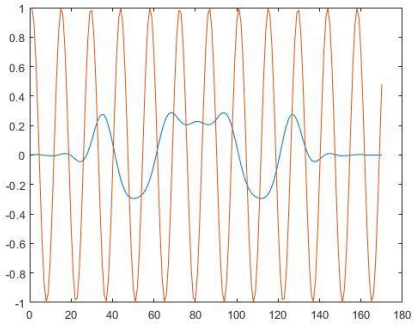
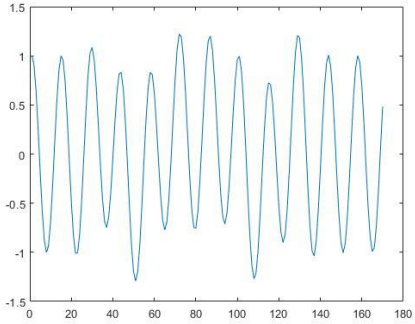
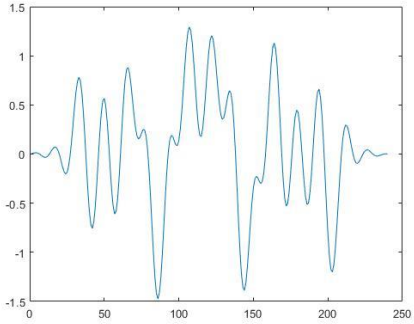
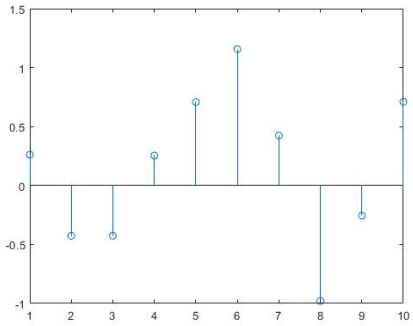
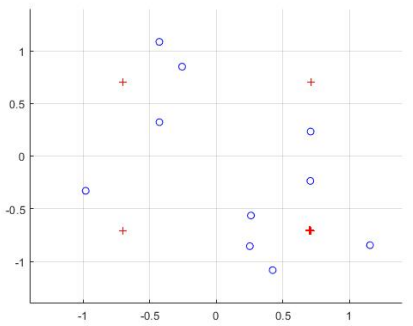
    The channel parameters, which is fixed for the generation
    - Sampling rate = 200 MHz
    - Maximum Doppler shift of diffuse components = 40 KHz
    - Discrete delays of 5-path channel = [0 4.5 8.5 12 25] (ns)
    - Average path gains = [0 -2 -10 -13 -16] (dB)
    - K Factor = 4 (Linear ratio of specular power to diffuse power)
    - Doppler shift of specular component = 20 KHz
5- Block five
    a. Matched filter: receives the data and convolves it with the pulse shaping again to have the response of raised cosine filter.
    b. down-sample: transform data back to bit-form (also called decimation which helps reducing the sampling rate)
6- Block six
    a. BER calculation: compares the transmitted bits with the received bits to calculate the bit error rate (BER)

Another important point to note is that the data generated is taken after the down conversion step, so all the frequencies are normalized to the signal BW except the frequencies in the channel model, it will assume that the data has a sampling rate of 200MHz and accordingly it will create the channel, so if all values of time and frequency is adjusted to the high sample rate, it will have the same channel response of the normalized frequency. So, we choose it to be this way to be more comparable to the range of frequencies we want the system to work with.

It is important to note that the power of the jamming and the noise are both calculated in the channel not after receiving of the data.

For illustration this is an example of the data going through each block for I-phase of QPSK modulation with oversampling = 10 with root raised cosine pulse shape and tone jamming

Table 2- 3 Output of each block in the communication system block diagarm

1- After Constellation mapper	2- After Up-sampling
	
3- After pulse shaping	**4- After pulse shaping and Jamming**
	
5- After pulse shaping + Jamming	**6- After matched filter**
	
7- After down-sampling	**8- Back to signal space**
	

It is important to note that this is a deep learning aware data generation so each parameter in the communication link or the jammer node must go through one of the following operations

1- Choose a fixed value: for parameter that is either insignificant in its effect to the model or it is fixed for certain type of communication links such as :
    - Channel model
    - Pulse shaping
    - Oversampling ratio

2- Choose a random value: this will prevent the deep learning model from overfitting to values that most likely to change through run time such as
    - Data being transmitted
    - The start of the data in the frame
    - Frequency and phase of the jammer

3- Sweeping: this is done on the key parameters to observe its effect on the model accuracy
    - SNR levels
    - Jamming Power which we swept firstly as a separate parameter from the SNR and secondly we connected it with the SNR and swept the value of SJNR where jamming and noise have equal power

2.2.9 Jamming Signals categories

1- Spot jamming is concentrated power directed towards one or more channels or frequencies.
2- Barrage jamming is power spread over all frequencies or channels at the same time.

Barrage jamming can be modeled as a White Gaussian Noise but only activated in part of the frame hence it will need continuous monitoring of the spectrum to be able to detect it correctly.

All the spot jamming types in their core are a sinusoidal wave, which we modelled as follows then the frequency of this sinusoidal is manipulated to generate the jamming techniques used in this project

$$Tone = \cos\left(2\pi \times \frac{freq_{jamming}}{oversampling} \times n\right) + j \sin\left(2\pi \times \frac{freq_{jamming}}{oversampling} \times n + phase\right)$$

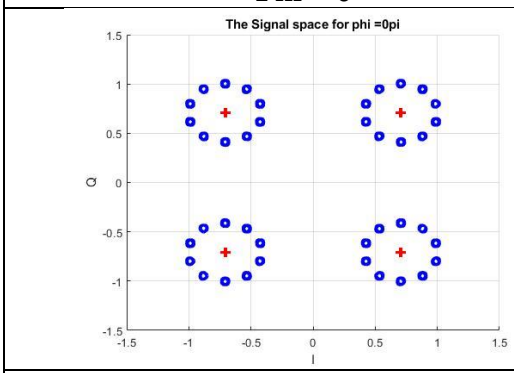
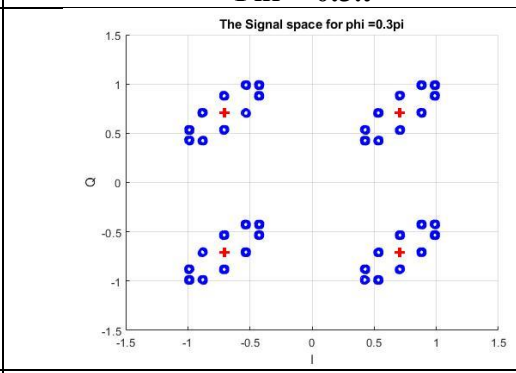
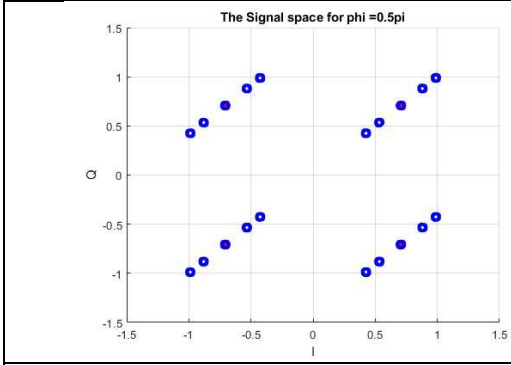
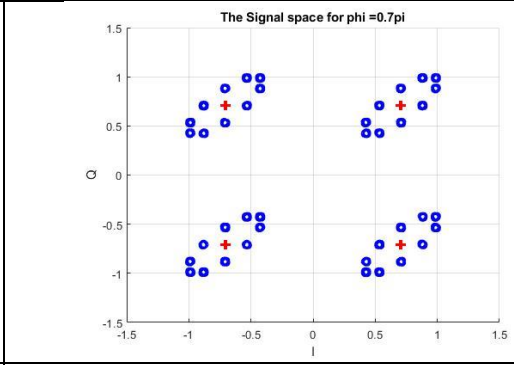
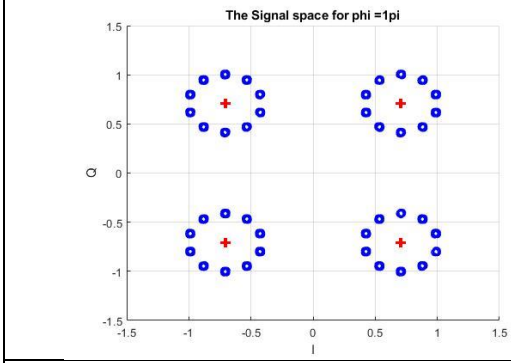
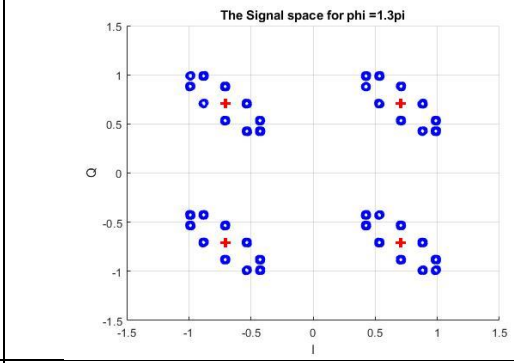
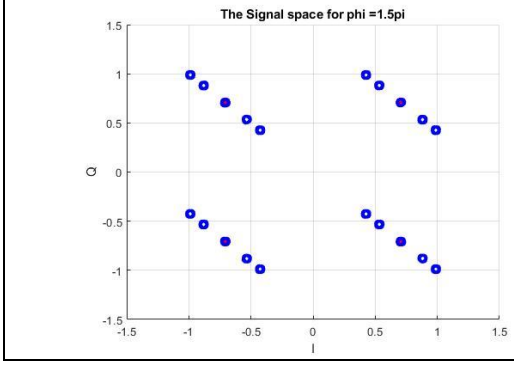
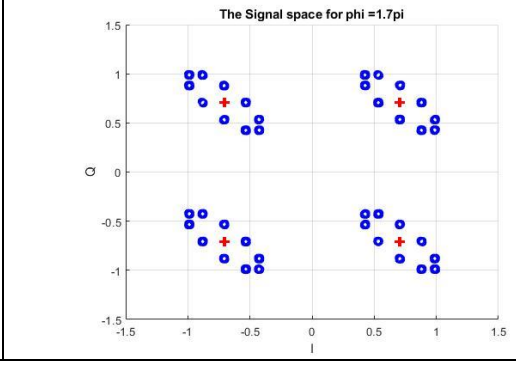
Two hyper-parameters here are:

- Phase difference between I-Q branches.
- Frequency of the tone normalized to the signal BW.

We will discuss the effect of each of them separately in the next pages.

Phase difference between I-Q branches. Table 2-4 shows the signal space of the QPSK modulation with tone jamming applied to it at different phase difference, the red points represent the ideal data of QPSK modulation only affected by the system span and filter shape and the blue points represent the resulting points when tone jamming is applied, then we can conclude that the effect of changing the phase difference between I and Q branches results in rotation of the blue points of the tone jamming.

Table 2- 4 Effect of phase difference between I and Q phases on the signal space of QPSK modulation

Phi = 0	**Phi = 0.3π**
	
Phi = 0.5π	**Phi = 0.7π**
	
Phi = π	**Phi = 1.3π**
	
Phi = 1.5π	**Phi = 1.7π**
	

Frequency of the tone normalized to the signal BW: as we can see in table 2-5 there are three things to notice:

1- The frequency of the jamming determines the distribution of the received points around the ideal point
2- As frequency increases the jamming power received decreases due to the matching filter and after the BW of the signal there is no effect of the jamming on the received data
3- At frequency = 0 the effect of the jammer is only a DC shift which is removed in the receiver directly using an AC coupling filer, so we will not generate data with zero frequency

Table 2- 5 Effect of tone frequency on the signal space of QPSK modulation

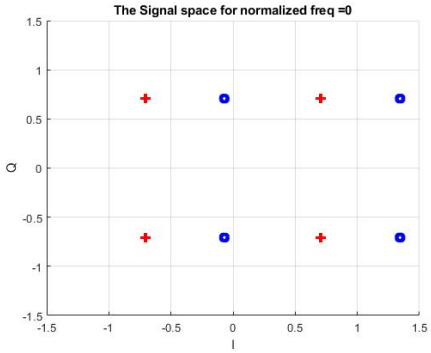
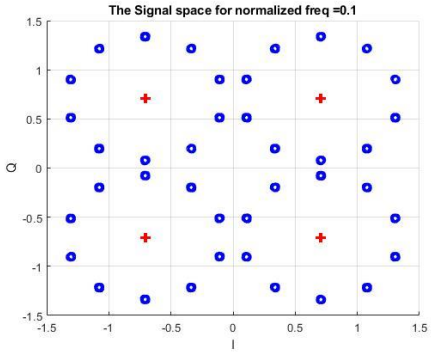
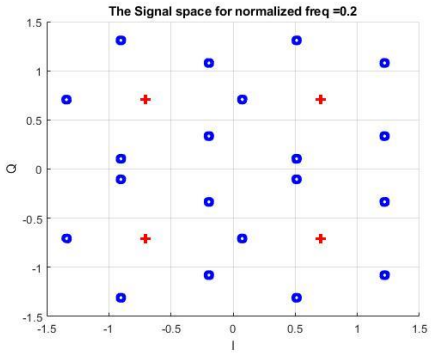
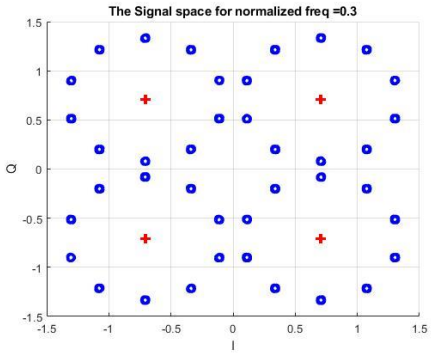
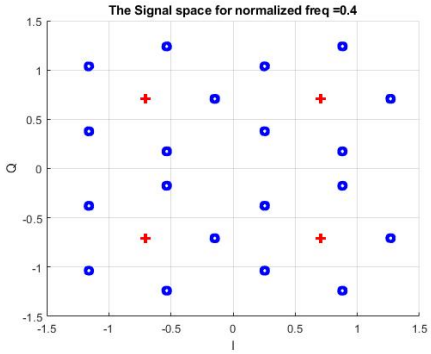
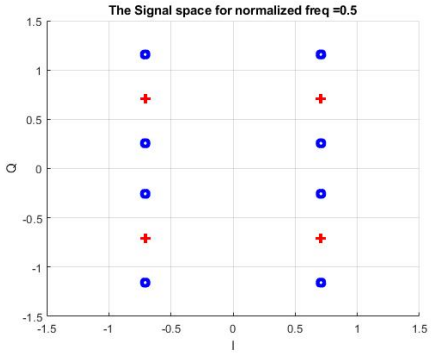
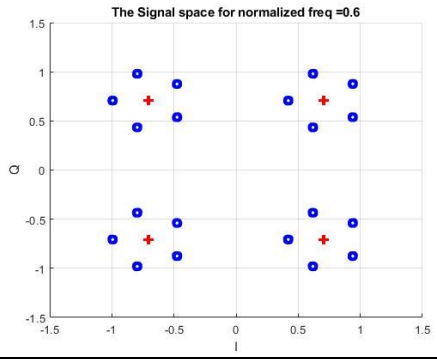
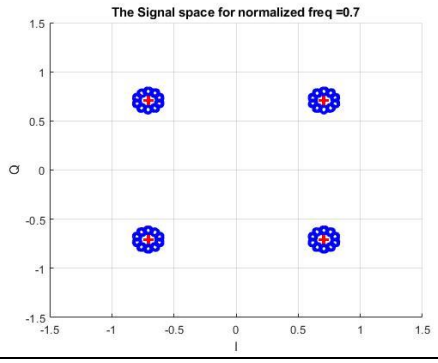
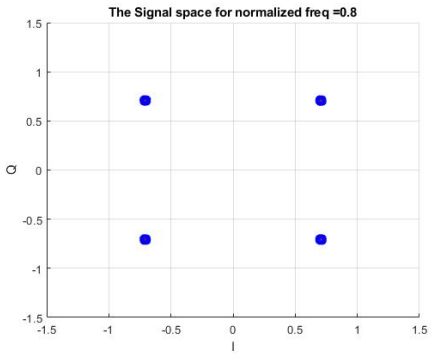
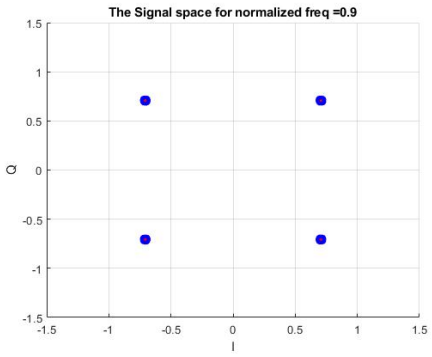
Freq = 0	**Freq = 0.1**
	
Freq = 0.2	**Freq = 0.3**
	
Freq = 0.4	**Freq = 0.5**
	

Table 2- 6 Effect of tone frequency on the signal space of QPSK modulation (continued)

Freq = 0.6	Freq = 0.7
	
Freq = 0.8	**Freq = 0.9**
	

2.2.10 The jamming types available in our dataset

Our jamming classification dataset consists of eight jamming techniques that lies under the two categories of jamming mentioned before; spot and barrage jamming, and it is important to note that all jammers are designed to give the same power if they are on all the time, so, any type of jamming that has a ON-OFF nature will give the frame less average jamming power compared to other types of jamming.

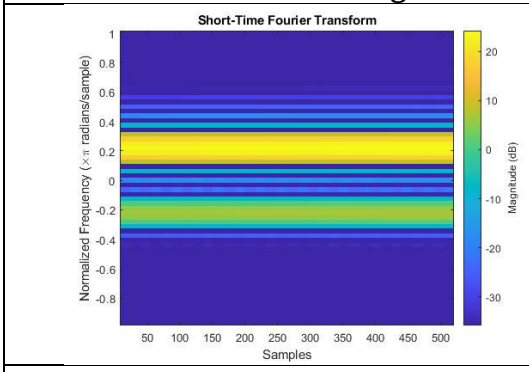
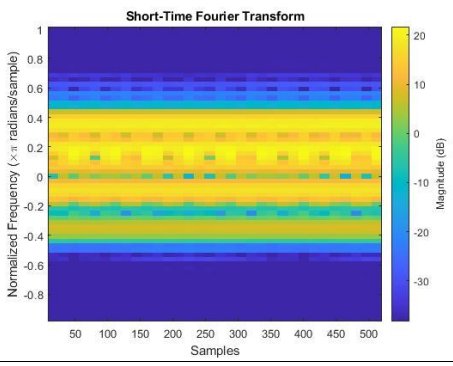
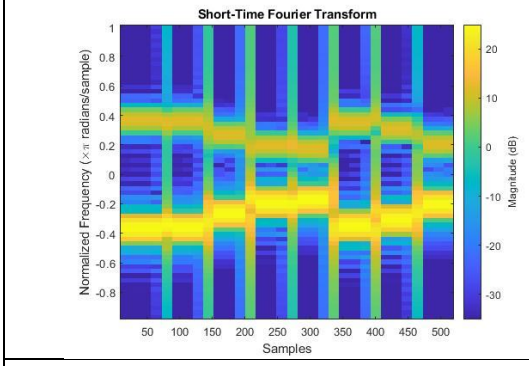
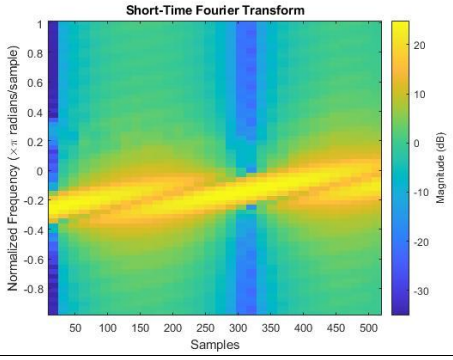
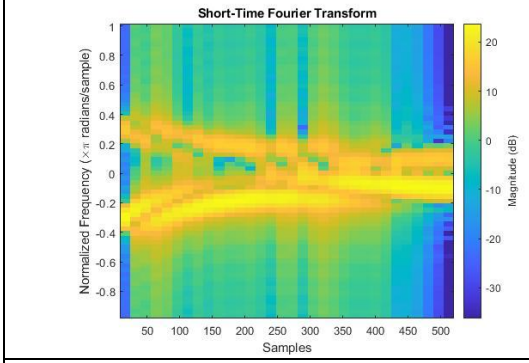
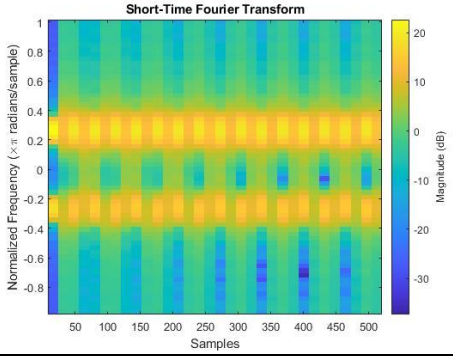
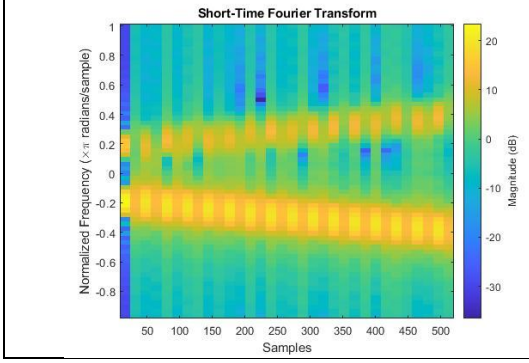
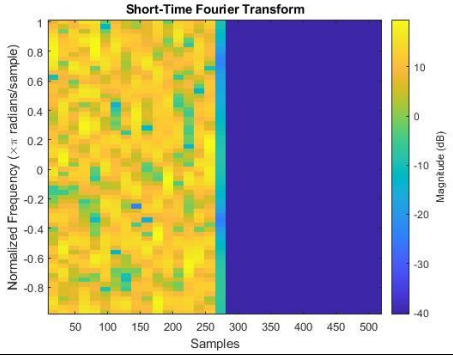
1- Tone jamming
   The simplest type of jamming which consists of a single tone jamming a certain frequency.
2- Multi-Tone jamming
   A type of jamming that is designed to distribute the jamming power across several frequencies by transmitting several tone jamming signals.
3- Hop Jamming
   A hop jamming is another type of spot jamming like tone jamming but the jamming frequency changes with time randomly.
4- Sweep Jamming
   A Sweep jamming is another type of spot jamming like tone jamming but it changes the jamming frequency linearly with time.
5- Parabolic Sweep Jamming
   A Parabolic Sweep jamming is another type of spot jamming like tone jamming but it changes the jamming frequency non-linearly with a third order function with time
6- ON-OFF Jamming
   A tone jamming that is powered on and off several times in the frame.
7- Chop & Interleave Jamming
   Chop & Interleave Jamming is a combination of sweep jamming and ON-OFF jamming.
8- Barrage Jamming
   A White Gaussian Noise added to a part of the frame.

For all the shown jamming techniques we have generated two groups of data, group one is a straight forward generation and group two is a more randomized version of it.

In table 2-7, we can see the STFT output of all the jamming types generated in our dataset:

Group one:

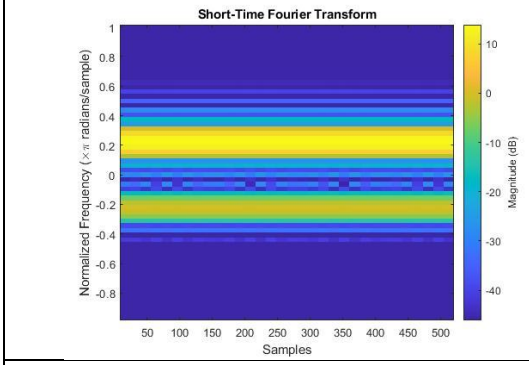
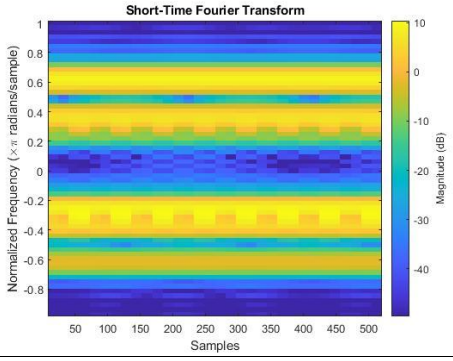
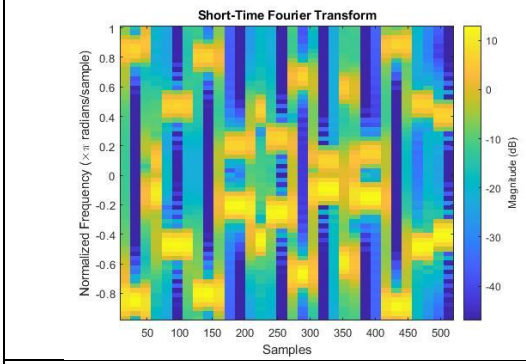
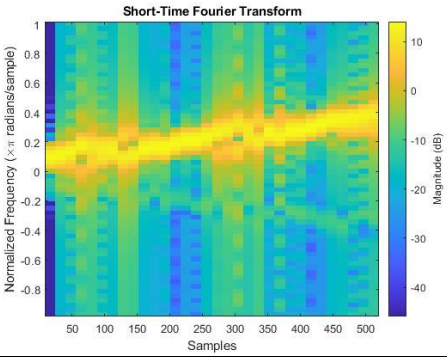
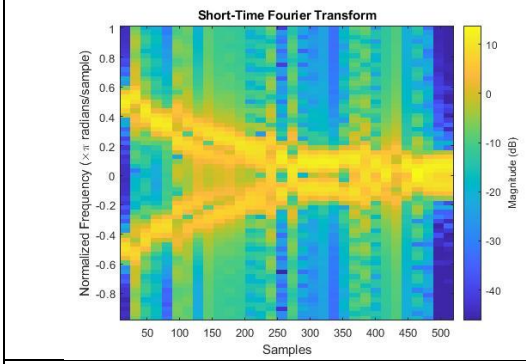
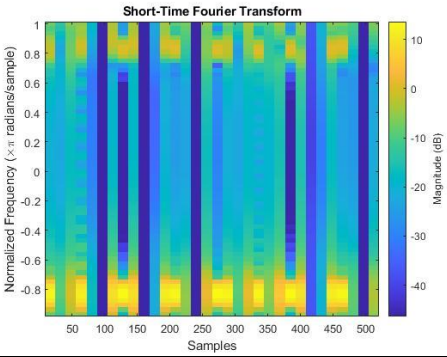
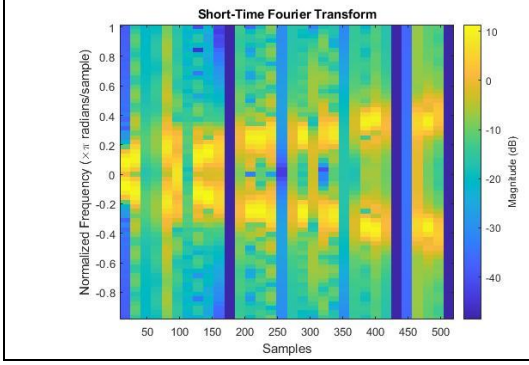
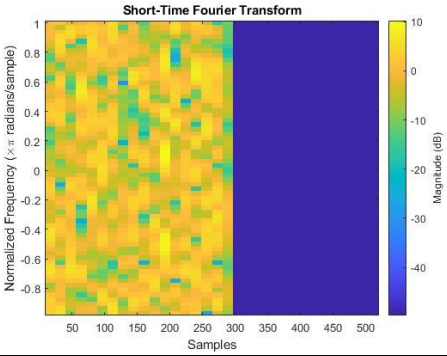
Table 2- 7 the STFT output of the jamming types generated in the dataset in group one

Tone Jamming	Multi-Tone Jamming
	
Hop Jamming	**Sweep Jamming**
	
Parabolic Sweep Jamming	**ON-OFF Jamming**
	
Chop & Interleave Jamming	**Barrage Jamming**
	

In table 2-8, we can see the STFT output of all the jamming types generated in our dataset:

Group two:

Table 2- 8 the STFT output of the jamming types generated in the dataset in group two

Tone Jamming	**Multi-Tone Jamming**
	
Hop Jamming	**Sweep Jamming**
	
Parabolic Sweep Jamming	**ON-OFF Jamming**
	
Chop & Interleave Jamming	**Barrage Jamming**
	

2.2.10.1 Group 2 specifications

1- Tone jamming

- Normalized random frequency from 0.1 to 2
- random phase random between I and Q phases from 0 to 2π

2- Multi-Tone jamming

- Normalized random frequency from 0.1 to 2 for each tone
- random phase random between I and Q phases from 0 to 2π for each tone
- random number of tones from 2 to 5 tones

3- Hop Jamming

- Normalized random frequency from 0.1 to 2 at each hop
- random phase random between I and Q phases from 0 to 2π at each hop
- Random holding period 15 to 50 points for each hop
- random start from 0 to 50 points

4- Sweep Jamming

- Normalized random start frequency from 0.1 to 0.5
- random phase random between I and Q phases from 0 to 2π
- random start from 0 to 25 points
- sweep window random from 10 to 25 points for each step
- frequency sweep range from 0.5 Hz to 1Hz

5- Parabolic Sweep Jamming

- Normalized random start frequency from 0.1 to 0.5
- random phase random between I and Q phases from 0 to 2π
- random start from 0 to 25 points
- sweep window random from 10 to 25 points for each step
- frequency sweep range from 0.5 Hz to 1Hz

6- ON-OFF Jamming

- Normalized random frequency from 0.1 to 2
- random phase random between I and Q phases from 0 to 2π
- random holding period from15 to 50 points for each step
- random start from 0 to 50 points

7- Chop & Interleave Jamming

- inherited from sweep jammer
- random holding period from15 to 50 points for each step
- random start from 0 to 50 points

8- Barrage Jamming

- stop point randomly chosen for each frame between the points from 170 to 341

2.3 Generated Datasets

2.3.1 Datasets Distributions

1- Distribution #1
   - ❖ one modulation technique
     - ('QPSK')
   - ❖ tone jamming only
   - ❖ AWGN channel

2- Distribution #2
   - ❖ five modulation techniques:
     - ('BPSK' , 'QPSK' , '16QAM' , '64QAM', 4PAM')
   - ❖ tone jamming only
   - ❖ AWGN channel

3- Distribution #3
   - ❖ five modulation techniques:
     - ('BPSK' , 'QPSK' , '16QAM' , '64QAM' , '4PAM')
   - ❖ Four jamming types:
     - ('Tone' , 'Sweep' , 'Barrage' , 'Hop')
   - ❖ With AWGN channel

4- Distribution #4
   - ❖ five modulation Techniques:
     - ('BPSK' , 'QPSK' , '16QAM' , '64QAM' , '4PAM')
   - ❖ Four jamming types:
     - ('Tone' , 'Sweep' , 'Barrage' , 'Hop')
   - ❖ With fading channel

5- Distribution #5
   - ❖ Adding four new techniques:
     - (‘8PAM' , '8PSK' , '16PSK' , '256QAM')
   - ❖ Adding four jamming types:
     - ('Multi-Tone' , ’Parabolic-Sweep' , 'On-Off' , 'C&I')
   - ❖ with fading channel

2.3.2 Tuning done in the dataset

We have generated many datasets in order to tune the system parameters such as frame size and oversampling ratio and the level of randomization that the deep learning can handle as we see in group one and two in the tables [2-7, 2-8].

So we found that the best value for the frame size is 128 symbol and we chose the oversampling ratio to be 4 in order to not increase the data size significantly so the total number of samples in the frame is 128×4 = 512 sample. We also found that the model can still handle the maximum randomization we generated in group two with only around 3% ~ 5% accuracy degrading.

2.3.3 Final Datasets

We have chosen some of the datasets we generated to be published and they are as follows:

1- A modulation classification dataset that belongs to Distribution #5 (before matched filter)
2- A Jamming classification dataset with jamming power = 0dB that belongs to Distribution #5 (before matched filter)
3- A Jamming classification dataset with SJNR from -20dB to 18dB with jamming power equal to the noise power that belongs to Distribution #5 (Before matched filter)
4- A Jamming classification dataset on empty channel (without data) with JNR from -20dB to 18dB that belongs to Distribution #5 (Before matched filter)

2.4 Communication and dataset generation conclusion

As we discussed through this literature there were so many distributions of the data for both the problem of modulation and jamming classification and many versions, some of them was generated for illustration or testing, and some of them generated for tuning some parameters of the communication system to achieve the highest accuracy with the most difficult parameters; as our goal is not just to implement the system but we needed to test the deep learning algorithm limits in the two problems.

Chapter 3: Deep Learning Part

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 so called neural networks hence the name artificial neural networks. Deep learning is implemented through neural networks architecture so it is also called a deep neural network.

The purpose of artificial neural networks (ANN) is to achieve a very simplified model of the human brain. By having the artificial neural networks try to learn tasks by mimicking the brain's behaviour. Like the brain, ANN also consists of a large set of neurons, which are a specialized cell elements. These neurons are activated in response to the input, the activation of the neurons allows the network to detect and classify the patterns. Depending on certain input data, a neural network will try to calculate the probability that the data belong to a certain class (e.g., an object in a specific image). The neural network can be trained to recognize different classes by providing it with a set of labelled training data, which is called supervised learning, it can also use unlabelled data in training which is called unsupervised learning. There are other types of learning it can use like semi-supervised learning and reinforcement learning, etc.

It is also the primary technology behind self-driving cars, speech recognition, image recognition, automatic machine translation, etc.

3.1.2 Convolutional Neural Networks Overview:

Convolutional Neural Networks (CNNs) are a special type of Neural Networks that provide an additional spatial property to ANN, they are commonly used with visual data, and have shown an outstanding performance on various benchmarks. CNN provide a powerful learning ability due to using multiple feature extraction stages (filters), each layer has filters that extract some information then pass them to the next layer, this information can be as simple as edge detection in the first layers and become more complex through the layers that can automatically learn representations from the data and can reach to detect faces, cars, etc. in the end see figure 3-1. This automatic feature extraction and learning without human involvement is what make the CNN in particular and Deep Learning in general a very widely-known powerful technique.

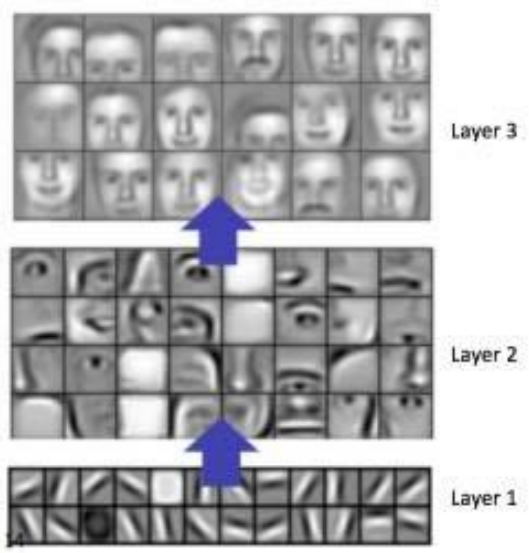


Figure 3- 1 Face detection enhances as undergoes several convolutional operations.

The CNN is divided into multiple learning stages through a set of convolutional layers, non-linear processing units (like ReLU), and subsampling layers (pooling). Each layer performs multiple transformations using a bank of convolutional kernels (filters). Convolution operation extracts locally correlated features by dividing the image into small slices, making it capable of learning features. Output of the convolutional kernels is assigned to non-linear processing (ReLU) units, which not only helps in learning abstraction but also adds non-linearity in the feature space which reduces the linear dependency that can occur due to many consecutive layers if the non-linear units is not present, Output of the non-linear function (ReLU) is usually followed by subsampling (Pooling), which helps in reducing the output size, and also makes the input invariant to geometrical distortions.

3.1.3 Convolutional Neural Networks Layers:

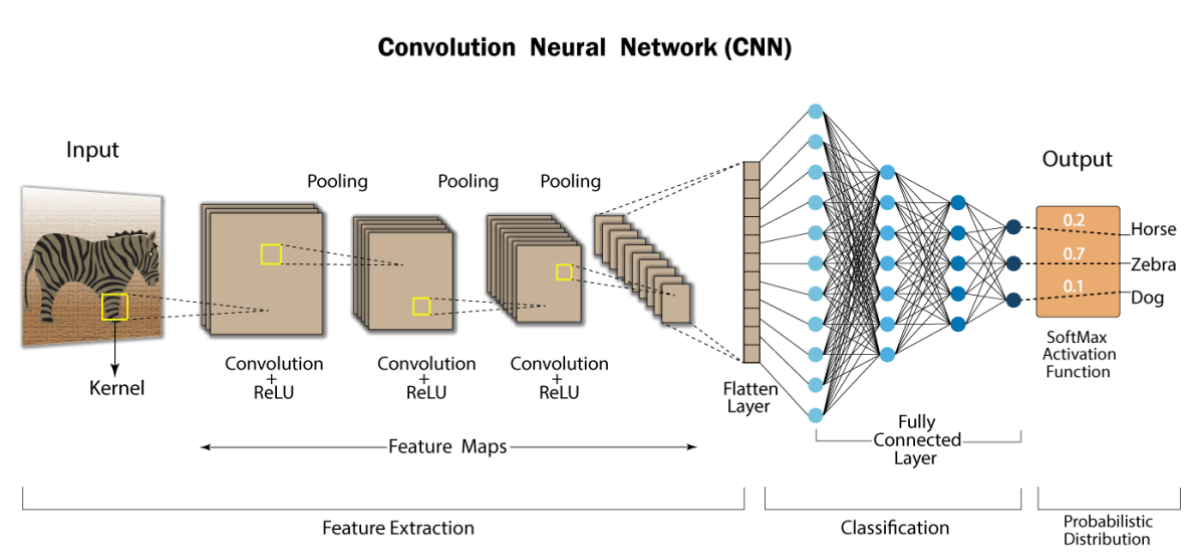


Figure 3- 2 Convolutional Neural Networks Layers

3.1.3.1 Convolutional Layer:

As in Figure 3-2, the first layer in a CNN is a Convolutional Layer which consists of set of trainable filters. The convolutional layer takes in patches of pixels and passes them through a filter, the filter (or kernel) is used to detect patterns in the pixels. The convolutional layer receives N feature maps as input; each input feature map is convolved by a shifting window with (k_H x k_W x n_D) filter where n_D is the depth of the image, and this will generate one element in one output feature map. If the input RGB image has dimensions (n_H x n_W x 3) and the filter has dimension of k x k x 3, then the convoluted output is of dimensions will be (n_H-k_H +1) x (n_W-k_W +1) where S is the stride of the shifting window, which is normally smaller than *K*. A total of *M* output feature maps, for M filters used, will form the set of input feature maps for the next convolutional layer as shown in Figure 3-3, convolutional layer detects low-level features of images i.e.: edges and colors, and by stacking a number of convolutional layers, the network hierarchically learns high-level features of the image.

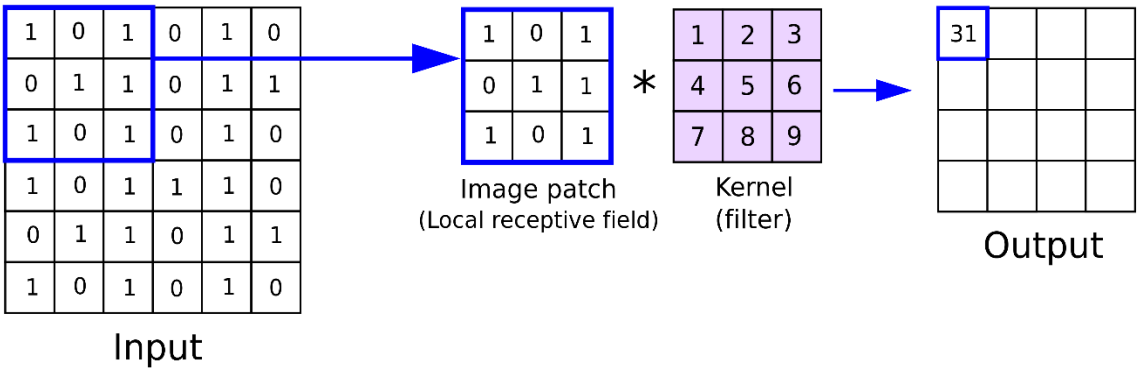


Figure 3- 3 Convolution Operation

3.1.3.2 Padding:

Padding is usually used to preserve the information that exists near the edge of the image, it is also used when concatenating the output of multiple Convolutional layers with different sizes as in insertion module; also it's used to preserve the original input size. No padding is called Valid Padding, and using padding to preserve the original input size is called Same Padding. The convoluted output is now of dimensions (n_H-k+2p+1) x (n_W-k+2p+1) as in Figure 3-4.

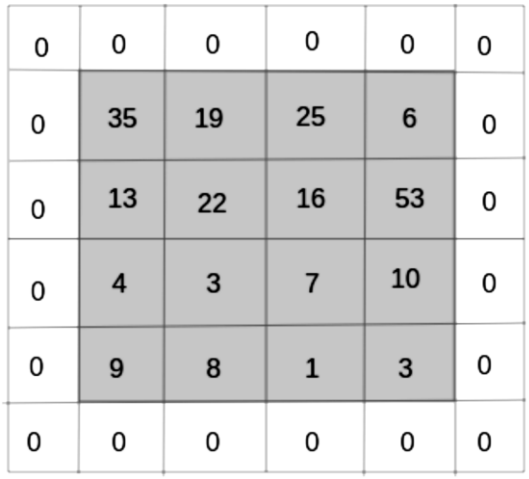


Figure 3- 4 Padding Operation

3.1.3.3 Stride:

Stride is the size of the step in which the filter moves with across the image until it reaches the upper right-hand corner. Stride is used mainly to decrease the output size so processing will be easier. The convoluted output is now of dimensions (($nH-k+2p$)/S +1) (($nw-k+2p$)/S+1) as the example for striding by 2 is presented in Figure 3-5.

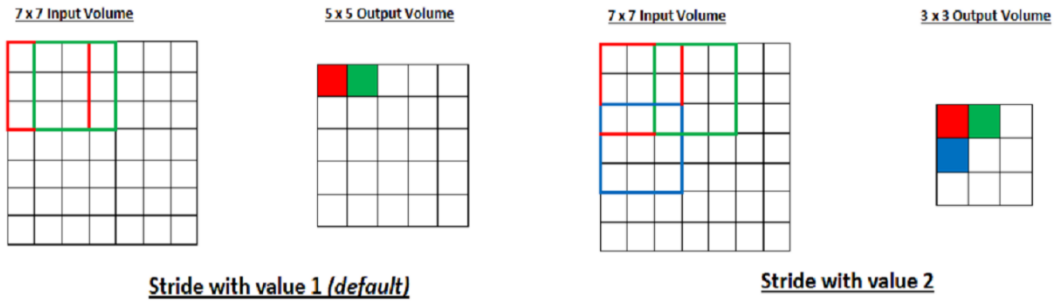


Figure 3- 5 Stride Operation

3.1.3.4 Pooling:

As in Figure 3-6, the Pooling layer (also called sub-sampling) reduce the dimensionality of each feature map of its input feature maps, the number of output feature maps is identical to that of input feature maps, while the dimensions of each feature map scale down according to the size of the sub-sampling window (called also kernel). There are three types of pooling, min pooling and it is very rare to use, and average pooling, and the most popular method the max pooling. The Max-pooling basically takes a filter $P \times P$ and a stride of length S, it then applies it to the input volume, and it returns the maximum number in every sub-region that the filter convolves around, and the average pooling do the same operation, but it returns the average of each sub-region not the maximum number, the average pooling acts like a smoothing filter to the image, but the max pooling selects the brighter pixels from the image, and it is better to identify the sharp features, as the average pooling may destroy them.

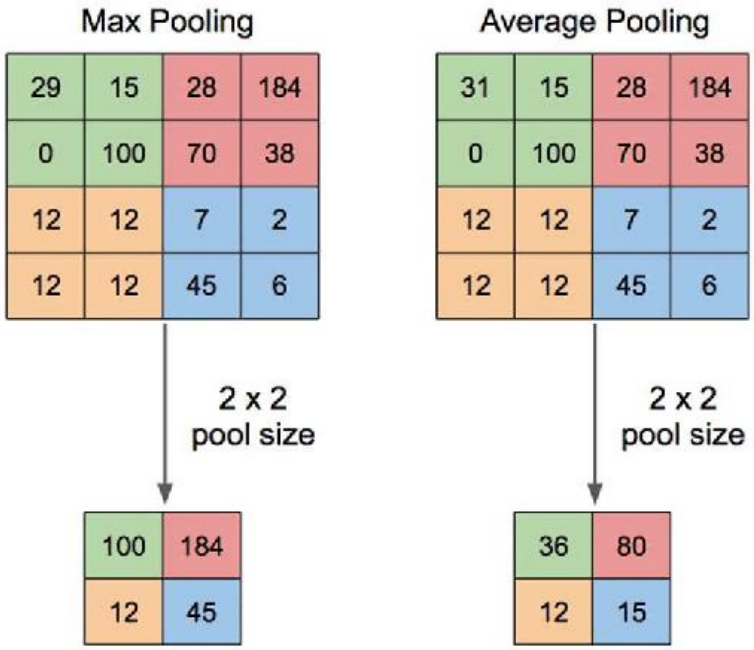


Figure 3- 6 Max and Average pooling operatoins

3.1.3.5 Fully Connected Layer (FC): ####### COPIED AS IT

The way this fully connected neural network layer (FC) works is that it looks at the output of the previous layer (which represent the activation maps of high level features) and determines which features most correlate to a particular class by unrolling the input features and the weights and multiply them and outputs an N dimensional vector where N is the number of classes .Also this layer is followed by soft max to show the most correlated class to the input as shown in Figure 2-7 which is an example for image classification.

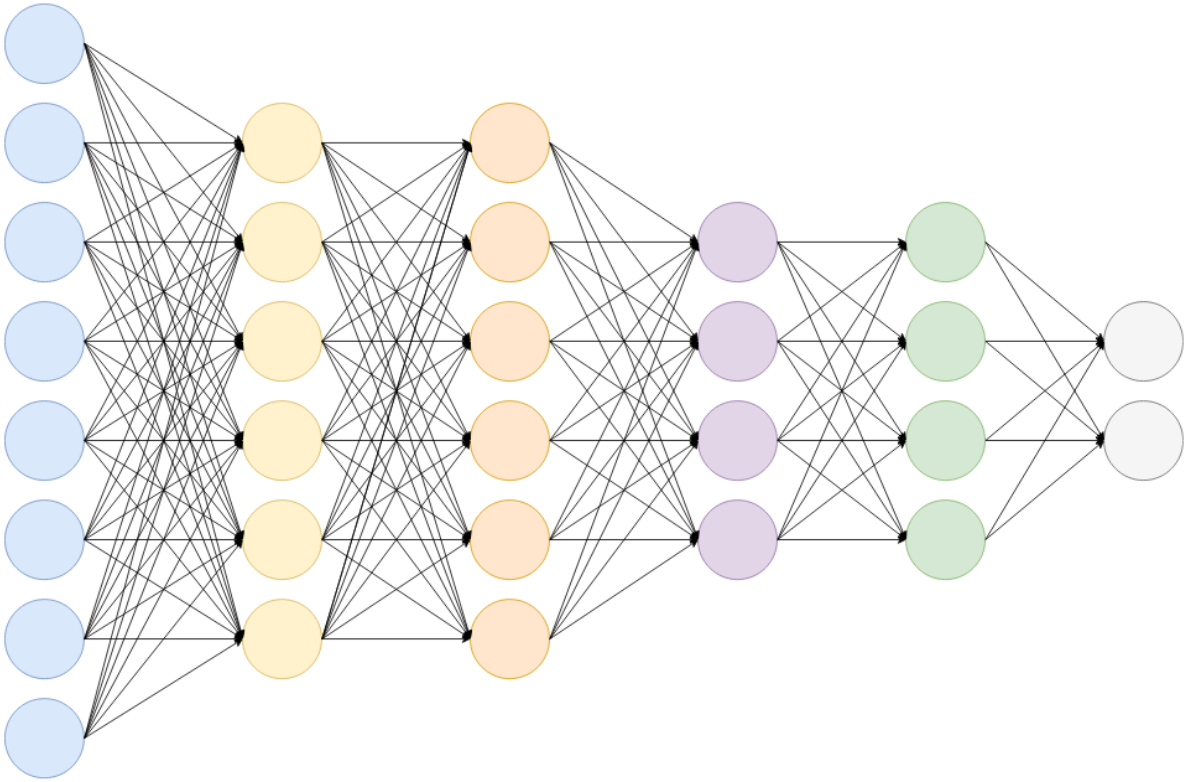


Figure 3- 7 Fully Connected Layer

3.1.3.6 Flatten:

The flatten operation converts the data into a 1-dimensional array in order to input it to the next layer. We flatten the output of the convolutional layers to create a single long 1-D vector; this vector contains all the features extracted by the convolution layers, the flattened vector is then passed to the final classification model that is the fully connected network. The flatten operation is described in Figure 3-8

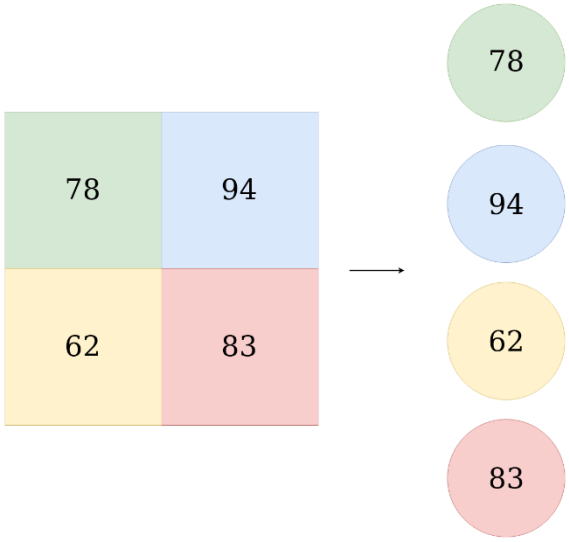


Figure 3- 8 Flatten operation

3.1.3.7: ReLU:

There are many activation functions to use such as sigmoid, Tanh, ReLU … etc. The activation function is a function that is applied after each layer; its purpose is to introduce a non-linearity to the layer output. If we don't add this non-linearity in NN, we can prove easily that all the NN layers can be reduced to one layer, so the activation functions are essential. The most widely used activation function is ReLU, the reason for its popularity is because it is relatively easy in implementation and provide a fast learning process compared to other functions. Its output is given by: $max\ (0, input)$, this gives us a linear relation in the positive half quarter of the 2-D grid, and zero in the negative half as shown in Figure 3-9.

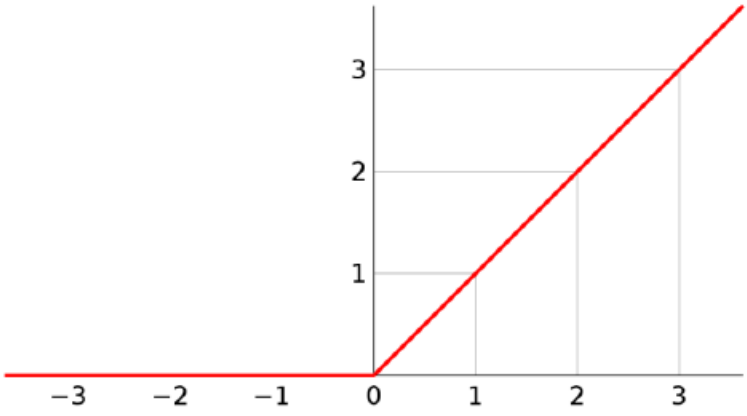


Figure 3- 9: The ReLU functoin

3.1.3.8: SoftMax:

SoftMax is also an activation function, it is used in the last layer in classification problems, its formula is as follows $Softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$. The function of this layer is to make the output in the form of a probability; meaning a number between (0 and 1) or a percentage from (0% to 100%). In the equation, we add the summation of all output nodes in the denominator, and each output node on the numerator to calculate the output. The output represents the probability of this class to be the correct one. The ideal case is one in the correct class and zero in other classes, and this is why we use the exponential function, as its nature as shown in Figure 3-10 makes the small numbers even smaller and the large numbers even larger; as it has a slow increasing rate at first for small numbers, but suddenly goes high for large numbers.

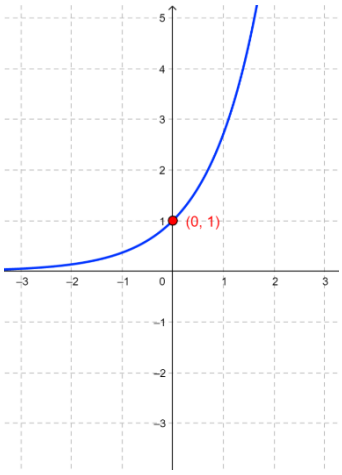


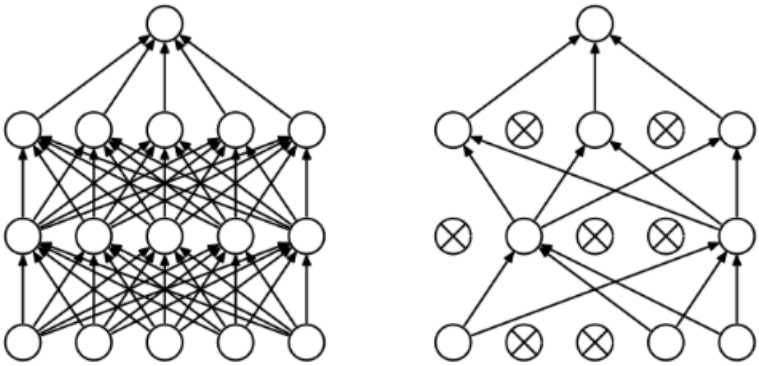
Figure 3- 10: Exponential function

3.1.3.9 Dropout:

The term "dropout" refers to dropping out units (both hidden and visible) in a neural network, as dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons, which is chosen at random. Calculations done by these units are no longer considered during a particular forward or backward path, and this is shown in the Figure 3-11.

We use the dropout to prevent over-fitting, and make the NN more immune to noise, as if some neurons are off, this makes the responsibility on the other neurons that are ON, and if we can use only these neurons while others are OFF to make the decision, this makes the model more immune to noise.

This dropout is used only while training, and after finishing training, all the neurons are on during inference, so we don't have to implement this in the digital part.



(a) Standard Neural Net (b) After applying dropout.

Figure 3- 11 Dropout process

3.1.3.10: Batch Normalization:

We use Batch normalization layer to make the mean of the output of this layer approaches zero and the variance approaches one, and we do this to solve the problem of vanishing gradients, that may occur in convolutional neural networks during training. It makes the convoluted value fall into the centre of the effective value region of the nonlinear function, so that vanishing gradient is avoided. The mean μ and variance σ^2 are as follows: $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i - \mu, and\ \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$, and the formula of the batch normalization is $Y = \frac{\gamma(x_i - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta$, where the parameters of this layer are the scaling factor γ , the translation factor β , the mean μ, the variance σ^2, and the added ϵ in the denominator is used to prevent the denominator from being 0, and the value of ϵ is 0.001 during training.

3.1.4 Pre-processing Techniques:

As mentioned in the problem statement our main concern here is anomaly detection by means of spectrum sensing, so the model needs to see the data in frequency domain. Therefore, in the upcoming subs-sections we will discuss some techniques used to transform the time series signals into frequency domain.

3.1.4.1 Fast Fourier Transform (FFT):

FFT provides frequency information about the signal; it represents the signal by its magnitude and frequency. The formula used to compute FFT is: $F(\omega) = \int_{-\infty}^{\infty} f(t) * e^{-i\omega t} dt.$

FFT can represent the signal in the frequency domain by telling us what are the frequency components that compose the signal, or in other words, what frequencies are present in the signal, but it has a limitation, that is, it can't tell us when in time those frequencies exist. In other words, the Fourier Transform is ideal for the stationary signals. The stationary signals are the signals that do not change with time, for example, a sine wave will always be a sine wave with the same frequency no matter the time, and this is not our case, as it is known that the signals in spectrum can change with time, so we had to find another technique that could tell us both the frequency component and when in time it exists.

3.1.4.2 Short time Fourier Transform (STFT):

The STFT was developed to overcome the poor time resolution of the Fourier Transform, it gives us a Time-Frequency representation of the signal. The methodology of this transform is to take window of the signal for a certain time period, then apply the Fourier Transform to it, as we can assume that a portion (the taken window) of the non-stationary signal is stationary, then shift that window, and apply Fourier Transform on the new portion and so on. This technique can tell us that the signal has some certain frequencies in this time window (or time frame), it is an enhancement of the FFT, as the FFT works on the whole signal at once. There are various types of windows such as a rectangular pulse window, a hanning window, a hamming window…, etc. We can have an overlap between the windows too, the Figure 3-11 describes the STFT operation. The window in the figure is a hanning window, and the overlap is because the hanning window approaches zero at its ends hence that part of the signal will be lost due to multiplication by zero, so to overcome this we allow overlapping between the windows as shown in the figure. The hanning window is commonly used because it provides good frequency resolution and leakage protection with fair amplitude accuracy.

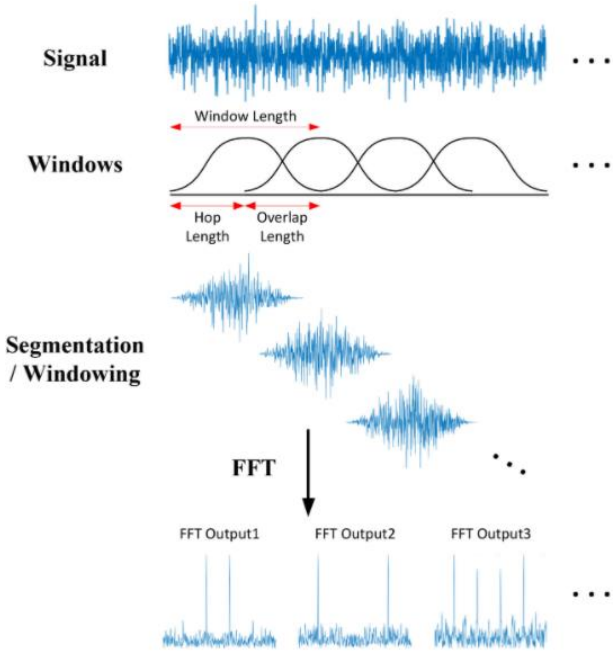


Figure 3- 12 STFT Operation

The limitation of STFT is that the window size is finite, which in turn reduces the frequency resolution, also, the window size is fixed which leads to a fixed frequency and time resolution. The relation between time resolution and frequency resolution is inverse proportionality, as when we use a wide window, it will give us a suitable time resolution, and will capture the low frequency changes as it happens in that wide time window, but it will give us a low frequency resolution, as it can't capture the high frequency changes, and vice versa.

The formula used to compute STFT is $F(\tau,\omega) = \int_{-\infty}^{\infty} f(t) * w(t-\tau) * e^{-i\omega t} dt$. When compared to the FFT formula in the previous section, the only difference between them is the added window function $w(t-\tau)$, where τ is the time localization variable.

3.1.4.3 The wavelet Transform:

This is another technique to transform the signal into the frequency domain, it is based on STFT yet it overcomes its drawbacks. We use a single shifting window in STFT but in wavelet transform we use a shifting window of multiple windows as in figure 3-12, each window has a function called basis function. This multi-window overcome the drawbacks in the STFT as some of these basis functions capture low frequency components, and others capture high frequency components, which gives us a better resolution in both time and frequency.

The formula used to compute it is $F(s,\tau) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} f(t) * \varphi(\frac{t-\tau}{s}) dt$. In comparison to the STFT formula, the window function here became basis functions φ, τ is the wavelet location, and s is a frequency scaling parameter.

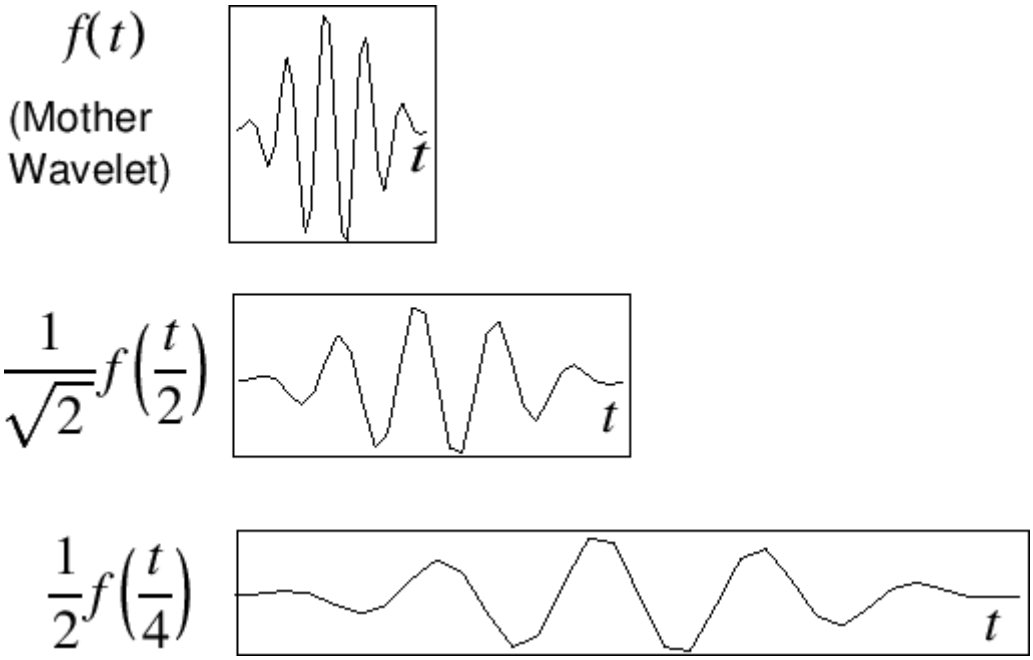


Figure 3- 13: example of the basis function of the wallet Transform

The limitation here is that the wavelet transform is relatively hard to implement, and because we are aiming for a real world application, and will deploy the model on FPGA, we decided not to use the wavelet transform. Instead, we are going to use STFT with a resendable value of window size.

3.2 Related work:

Under the umbrella of the spectrum sensing for anomaly detection, we searched for any related work, and we found the modulation detection problem that was described in the Deep Learning Modulation Recognition for RF Spectrum Monitoring paper, they used the PSI (Ψ) model that was developed by M. Kulin, the model architecture is described in Figure 3-13, the idea behind this is to see the data in different representations, as they transform the IQ data into frequency domain using FFT, and also use Amplitude and phase representation. Each representation goes through a separate convolutional neural network in its branch, they then pad the output of each branch and concatenate them, after doing a feature extraction in each branch, the output goes through a convolutional network. This was a brilliant idea, as each representation containing some information, and the CNN try to extract this information, and concatenate them to decide the final output.

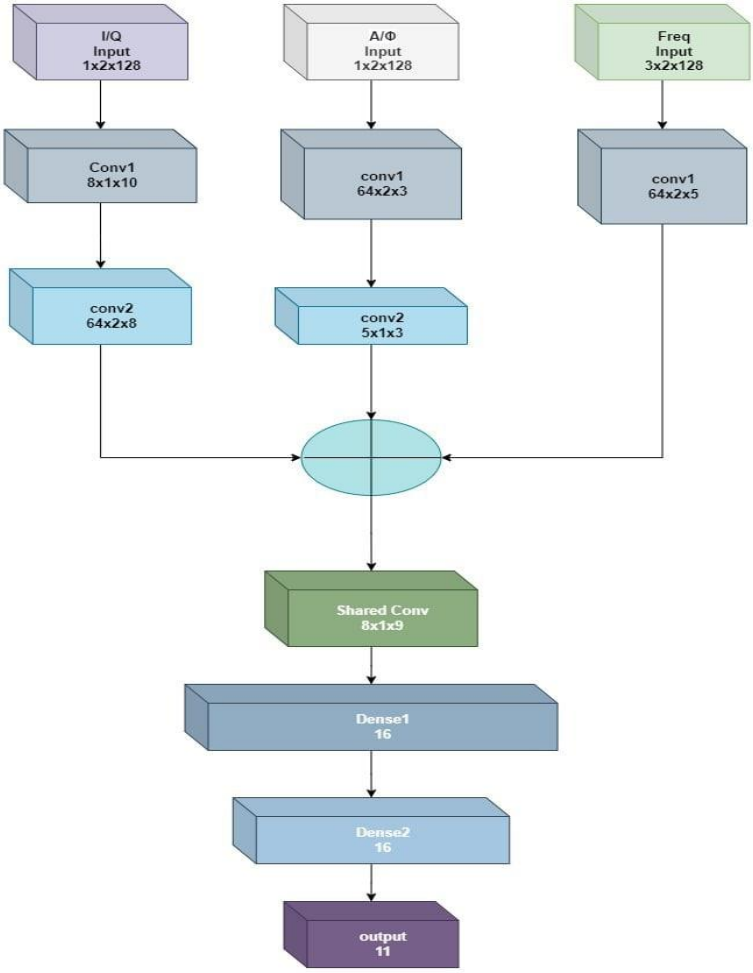


Figure 3-14 PSI (Ψ) model

We tried to replicated this model, and got the same result that was described in the paper, we used the same dataset, RADIOML 2016.10A, that is provided by DEEPSIG the dataset includes 11 modulation schemes 8 Digital and 3 analog modulation, They are : BPSK, QPSK, 8PSK, 16QAM, 64QAM, BFSK, CPFSK, PAM4 for digital modulations and WB-FM, AM-SSB, AM-DSB for analog modulation, the dataset consists of 220,000 In-phase and quadrature (I/Q) represented data vectors divided into 20 different SNRs limited between [-20 : 18] dB and each frame consists of 256 sample (2 x 128), and they use 70% of data is used for training and 30% of data is used for test.

The model containing 36,888 parameters, and they can achieve average accuracy up to 54.38%, 82% for medium SNR levels, and 84% average accuracy for high SNR levels.
The results of their work is as follows:

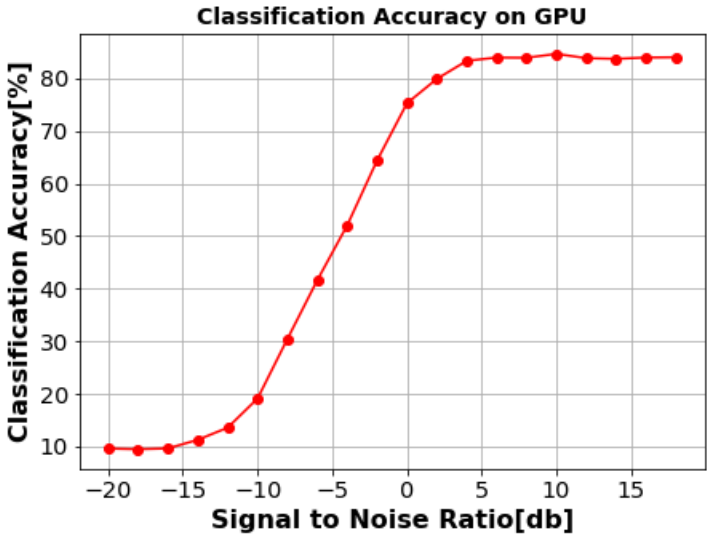


Figure 3- 15: Accuracy Vs. SNR

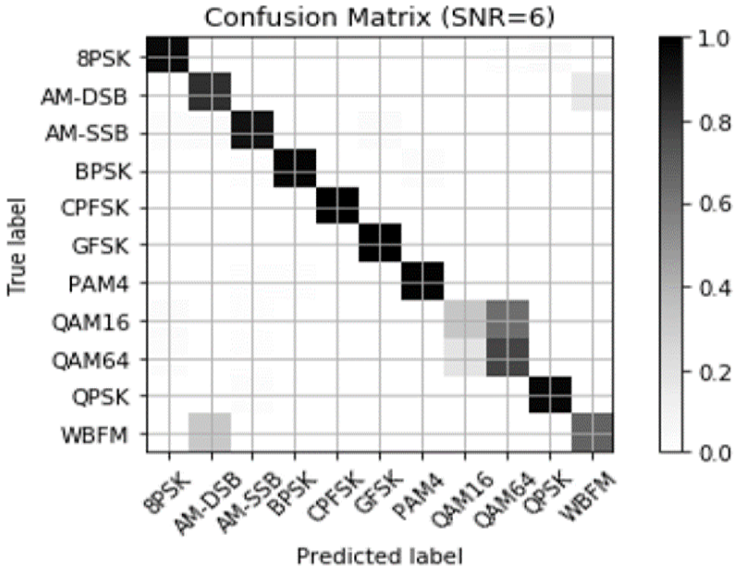


Figure 3- 16: Confusion Matrix at SNR=6

After learning more about the data, we made some changes to it as discussed in chapter two, the dataset was 128 pairs of IQ data, and with oversampling equal to 8, this mean it only containing 16 sample, and this is a very low number to detect the modulation scheme from, so we regenerated a dataset with the same channel parameters, and we made some changes, as we set the oversampling parameter to 4 and generated 512 pair of IQ signal with 128 samples, also, we would like to mention that we used only nine digital modulation schemes, as follows; BPSK, QPSK, 8-PSK, 16-PSK, 16-QAM, 64-QAM, 256-QAM, 4-PAM, 8-PAM.

We were able to obtain an average accuracy equal to 63.15%, and average accuracy equal to 96.08% above zero dB, this is a great improvement in the Mid and high SNR levels, and the accuracy Vs SNR is as follows:

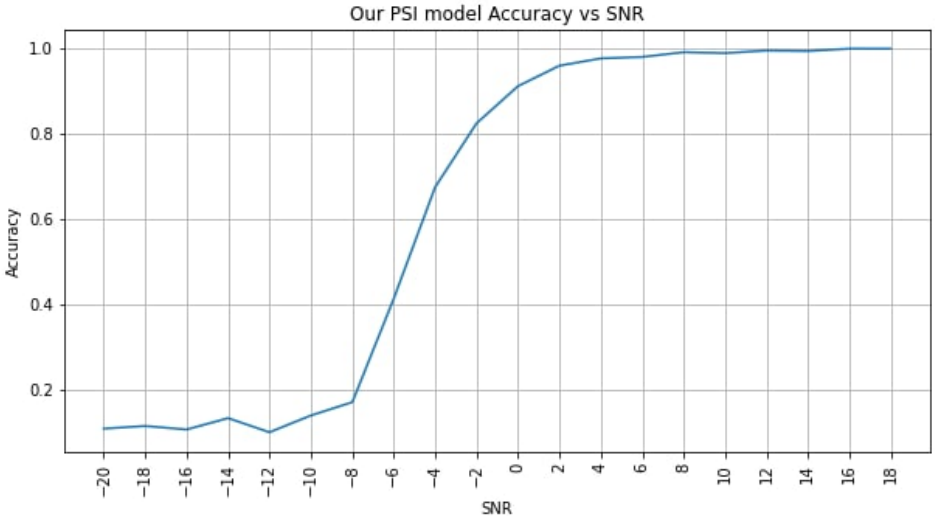


Figure 3- 17 Accuracy VS. SNR in our model using the new dataset

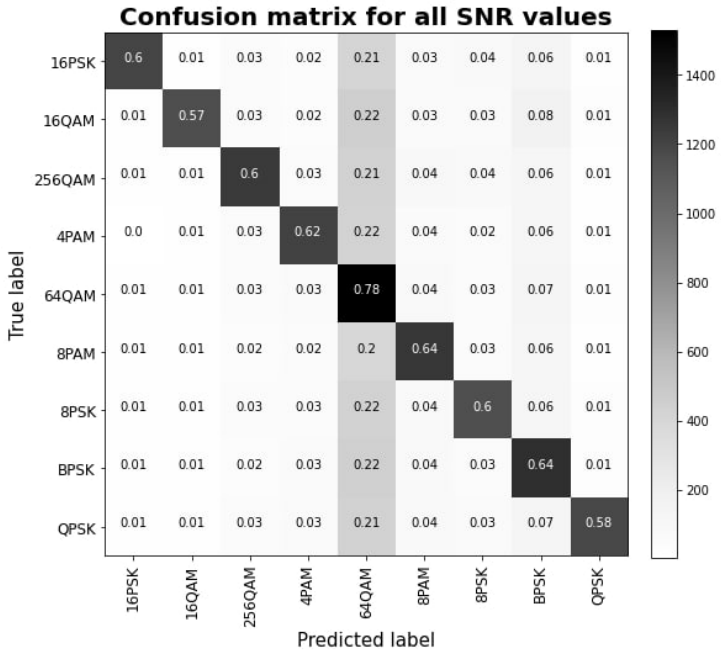


Figure 3- 18 Confusion matrix for all SNR values using the new dataset

And as we can see in figure 3-17, we reach almost 100% accuracy for high SNR levels, and we can obtain more than 90% at SNR level equal to zero dB.

3.3 Model Architectures

3.3.1 Overview of the work:

Our project statement is about jamming detection for wireless signals, but we choose to classify the jamming not just detect it, this is due to two main reasons, the first reason is that we we know the type of jamming, this will help us make an anti-jamming system, or we stop sending or receiving the data on that channel, and the other reason is that we have a plan, which is we want to make a full research on this case study, as we want to see if a certain parameter changed in the generation how would this affect the detection of a certain jamming type, and this was very obvious in the concussion matrix. Our aim is it to make the model work with the smallest number of parameters because it makes the model more immune to noise and data changes.

We built our models using TensorFlow platform, and we use numpy library in the pre-processing stage, we parse the data from the pickle file, and then split the data into a train set, a validation set, and a test set. Here, we use 85% of data as train, and 10% for validation, and 5% for test, we used Google Colab as an online platform to deploy our model, Colab provides us with a NVIDIA Tesla K80 GPU for free, and 16 GB of RAM, and we add our dataset on google drive, and connect both together.

3.3.2 Model 1:

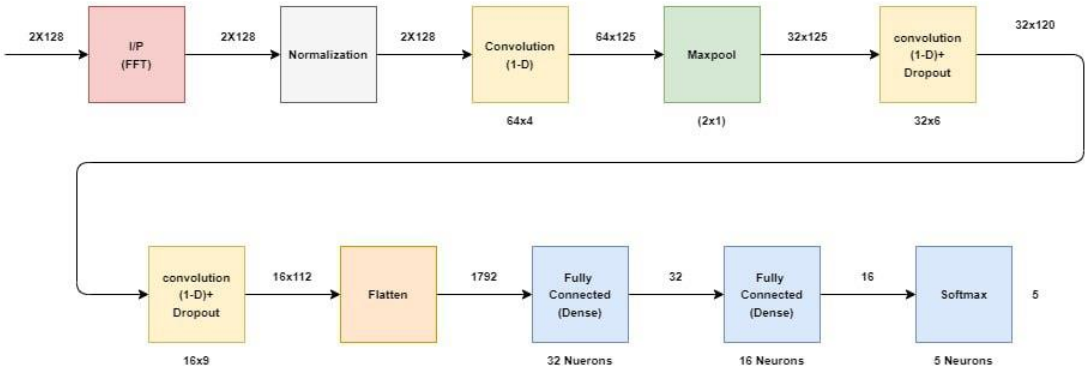


Figure 3- 19: First model Architecture

This model was developed on a data set from distribution #3 that we referred to in section 2.3.1, and the highlight specs in this data set were, it has an AWGN channel that was collected after a matched filter, and it has five modulation schemes: 'BPSK' , 'QPSK' , '16QAM' , '64QAM' , '4PAM'. It was developed to classify five classes; the Normal data, Barrage jamming, Hop jamming, Tone jamming, and finally Sweep jamming. The dataset consists of 100,000 pair of In-phase and quadrature (I/Q) represented data vectors, divided into 5 classes with 20,000 pair in each, and divided into 20 different SNR levels from -20dB to 18dB with 1000 pair in each level.

We decided to use FFT as a feature extraction function, this was inspired from the modulation detection model of the previous work. After we tried each data representation, we found that the frequency representation provides the highest accuracy, and IQ has the lowest accuracy, and we obtained these results, we achieved an average accuracy over all SNR levels equal to 77.40% and an average accuracy equal to 90.61% for SNR levels above 0dB. The model had a total number of parameters equal to 69,369.

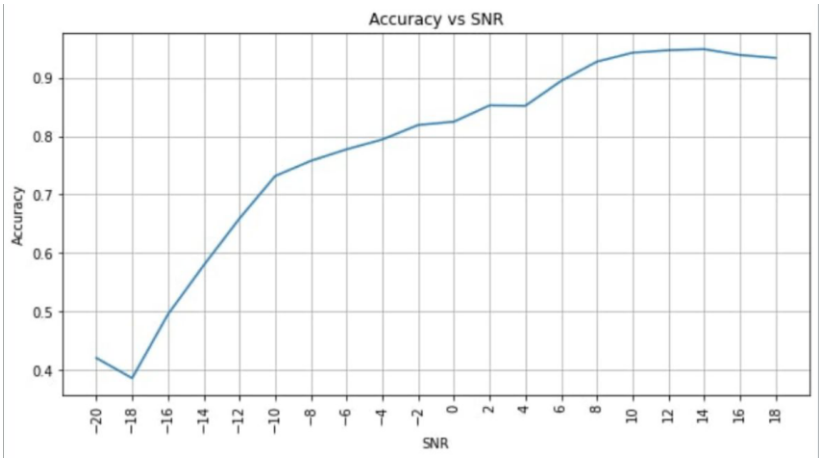


Figure 3- 20 Accuracy vs SNR for the jamming detection model using FFT

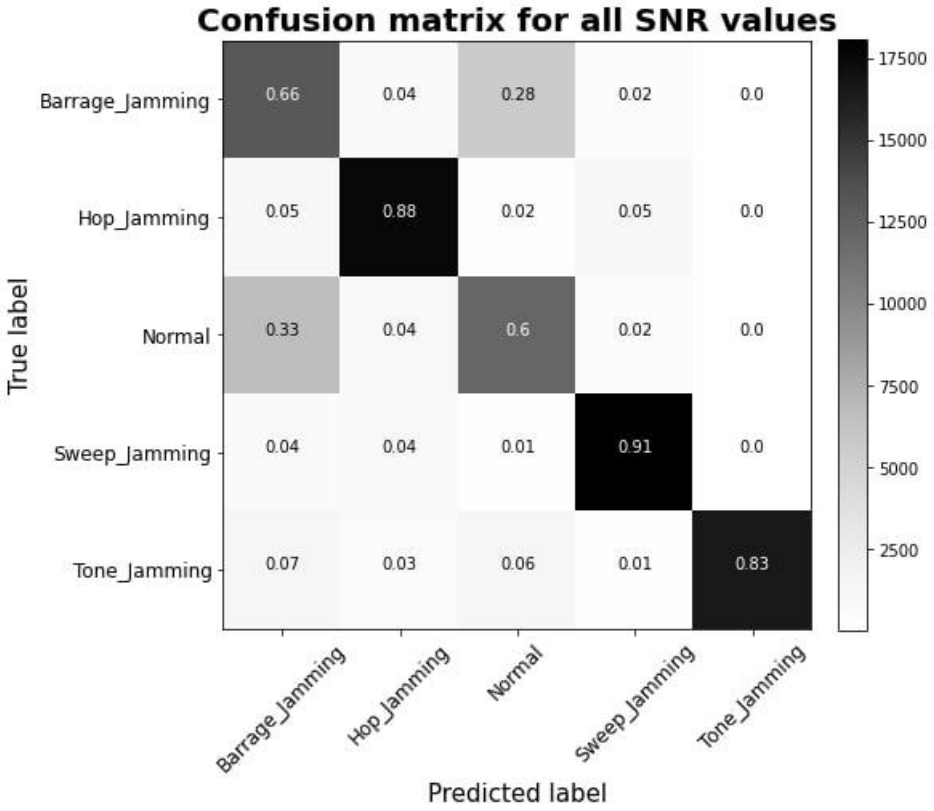


Figure 3- 21 Confusion matrix of Model 1 (FFT)

Due to the limitation we discussed in 3.1.4, we decided to use STFT from now on, and we use the same dataset, and using the same model architecture, we set the parameter of the STFT to be, the window size is 32, with a step of 16 point, and a 64 point FFT, when we used a window size smaller than the size of FFT, this will pad the window with zeroes to reach the size of FFT, after some investigation, we found that this way gives us more resolution for the output signal. STFT in Tensor-flow takes real numbers and returns complex numbers, so we separate the I-branch from the Q-branch, and apply STFT on both, then each of them becomes complex, we separate the real and imaginary, the final input shape was 31*33 and 4 channels, the total number of parameters was 13,637, and the final result for this model was an average accuracy equal to 76.56% for all SNR levels, and for signals above zero dB we can obtain average accuracy equal to 91.24%

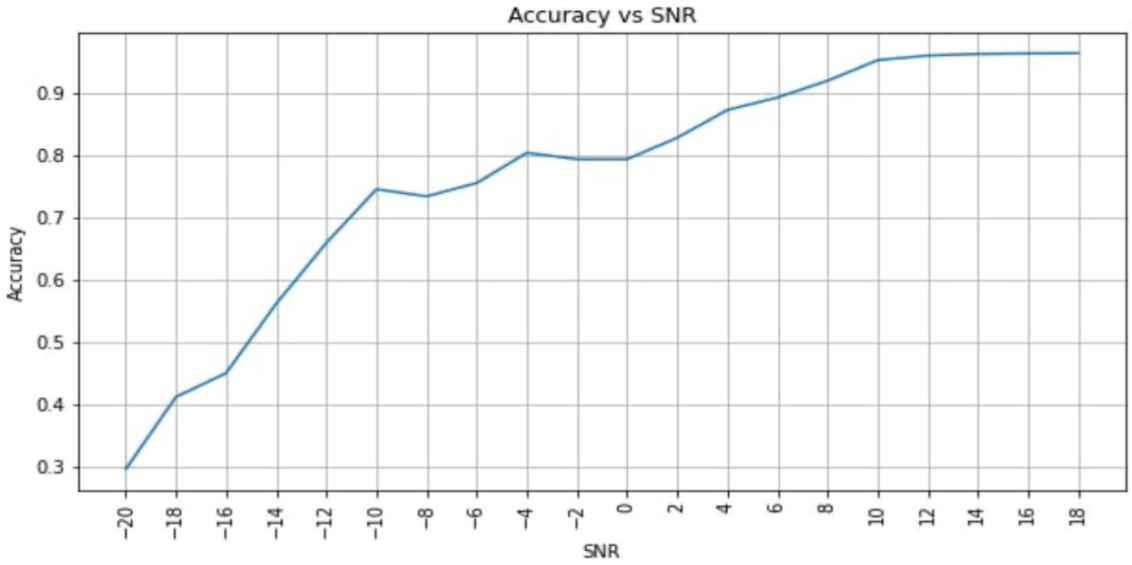


Figure 3- 22 Accuracy vs SNR for the jamming detection model using STFT

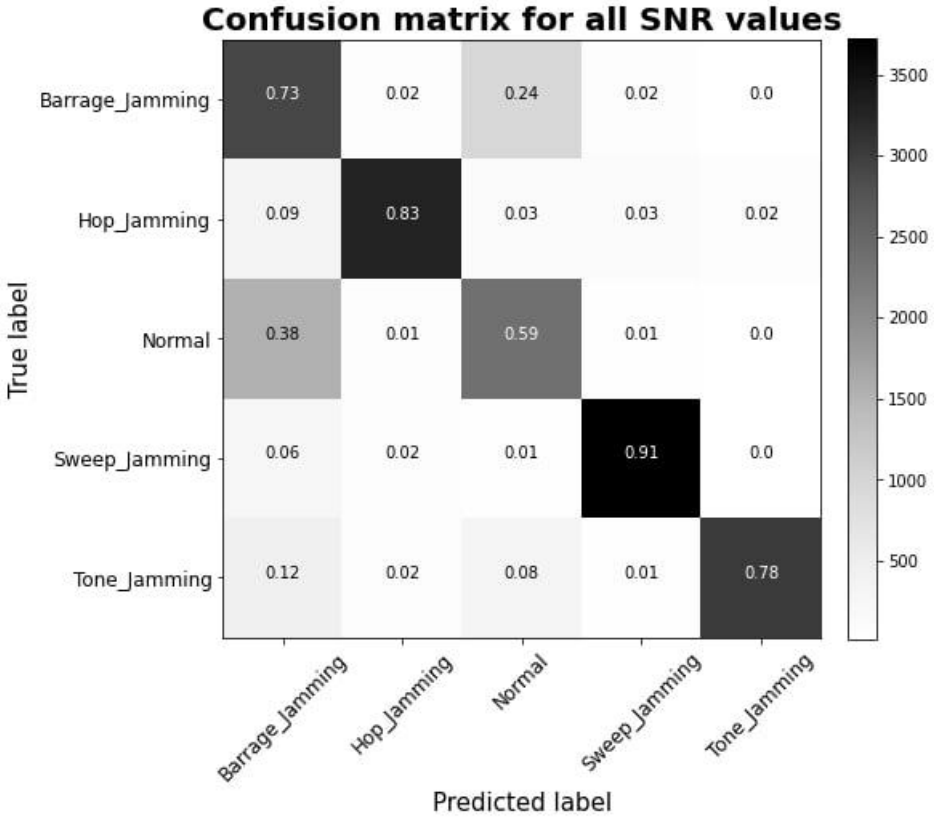


Figure 3- 23 Confusion matrix of Model 1 using STFT

3.3.3 Model 2:

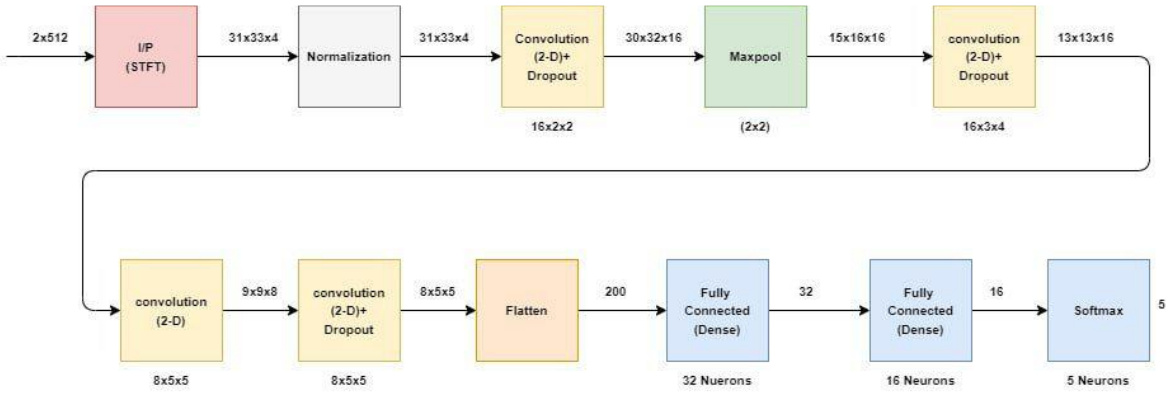


Figure 3- 24: Model 2 Architecture

And as our aim is to make jamming detection, and we don't know any information about the input signals, so we made a dataset before the matched filter, and this dataset was based on distribution#4 referred to in section 2.3.1, and from a quick look on the data's specs, it has a fading channel, and the same jamming types, and modulation techniques, the average model accuracy of all SNR levels is 62.17%, and average accuracy above 0db is 95.07%, and this drop in the accuracy is due to changing our point of view from after the matched filter to before the matched filter, it is more difficult to the model to detect the jamming. Figures 3-25, 3-26 shows the result of the model.

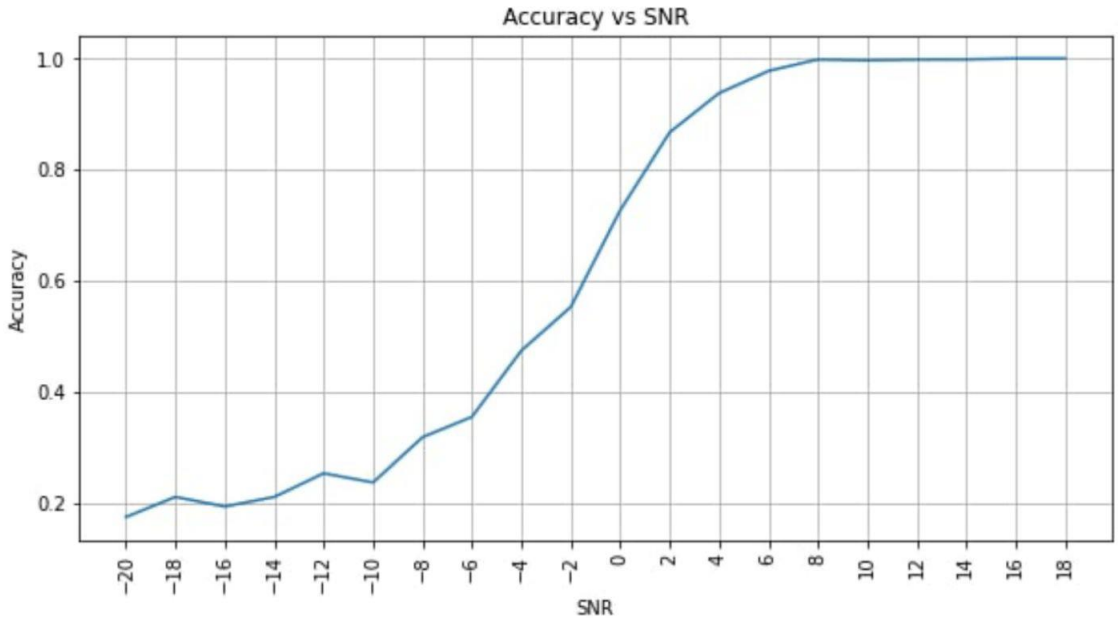


Figure 3- 25 Accuracy vs SNR for the modulation detection model

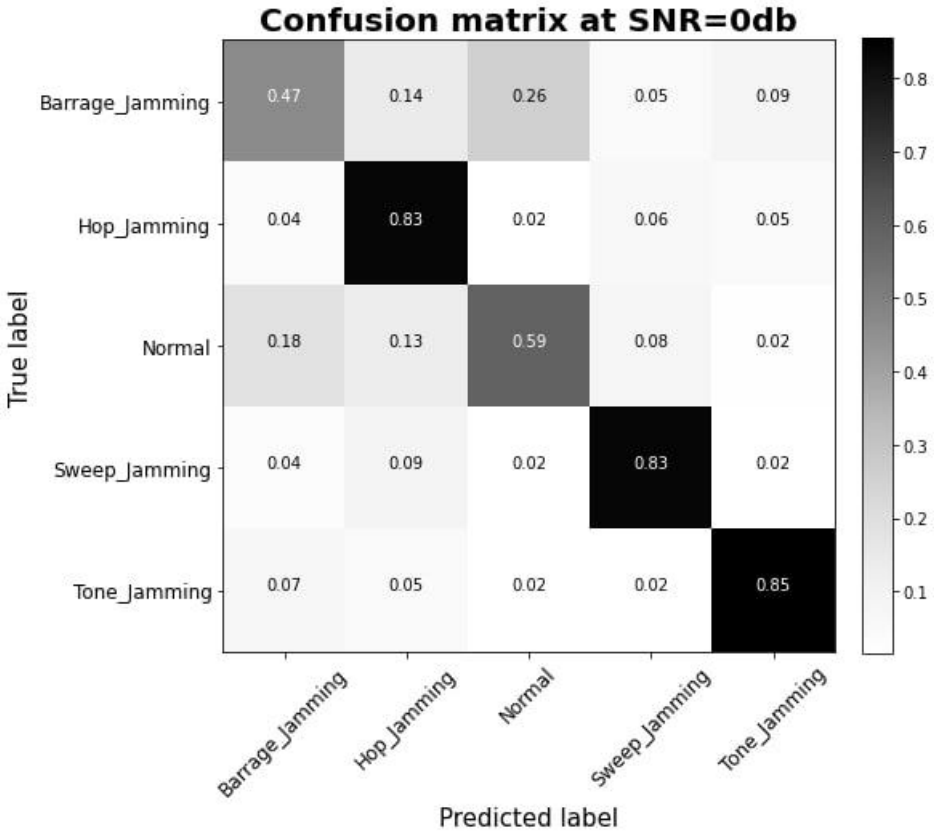


Figure 3- 26 Confusion matrix of Model 2 using STFT and distribution#4

3.3.4 Model 3:

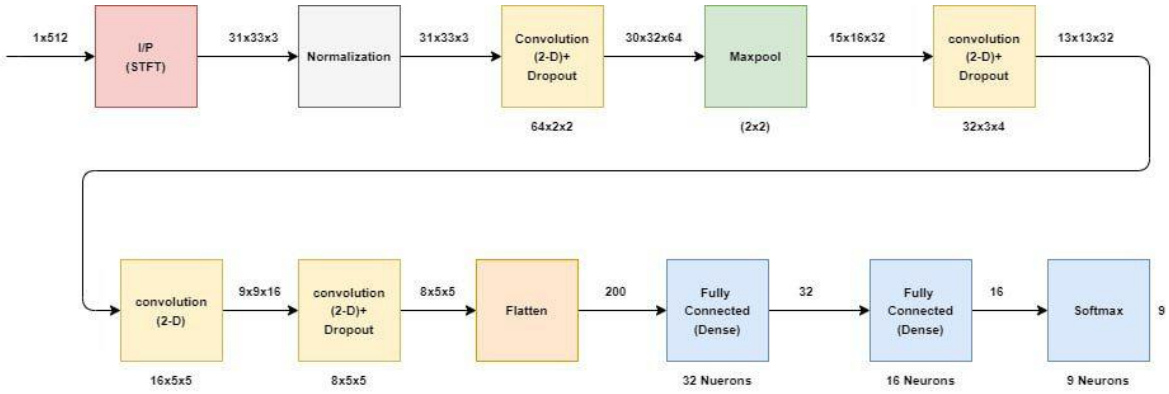


Figure 3- 27 Model 3 block diagram

In this model we change the input shape, as in previous models we separated the I branch and the Q branch, and processed them separately because STFT takes only a single real input, so in this model we took the magnitude of the I and Q branches of signal by adding their square values and then taking the square root of their addition, then input became 1*512, and the output was real part in a channel, and the imaginary part in the other channel, and we added the magnitude of them to the last channel, this made a jump in accuracy and improved the average accuracy by around 6%.

The dataset here was based on distribution#5 missioned in section 2.3.1, the dataset has the five modulation techniques as previous models, but we added more four jamming types; C&I jamming, Multi-Hop jamming, On-Off jamming, and Parabolic-Sweep jamming, by adding

more jamming types we expect the accuracy to drop, and this happened in the low SNR levels as the model guessed randomly which class it was, but for the middle and high SNR levels, we hit almost 100%.

We also changed the number of filters in the model, this change and the input shape change enabled us to reach **65.53%** average accuracy over all SNR levels, and **99.38%** for SNR higher than zero dB. Figures 3-28, 3-29, 3-30 shows the result of the model.

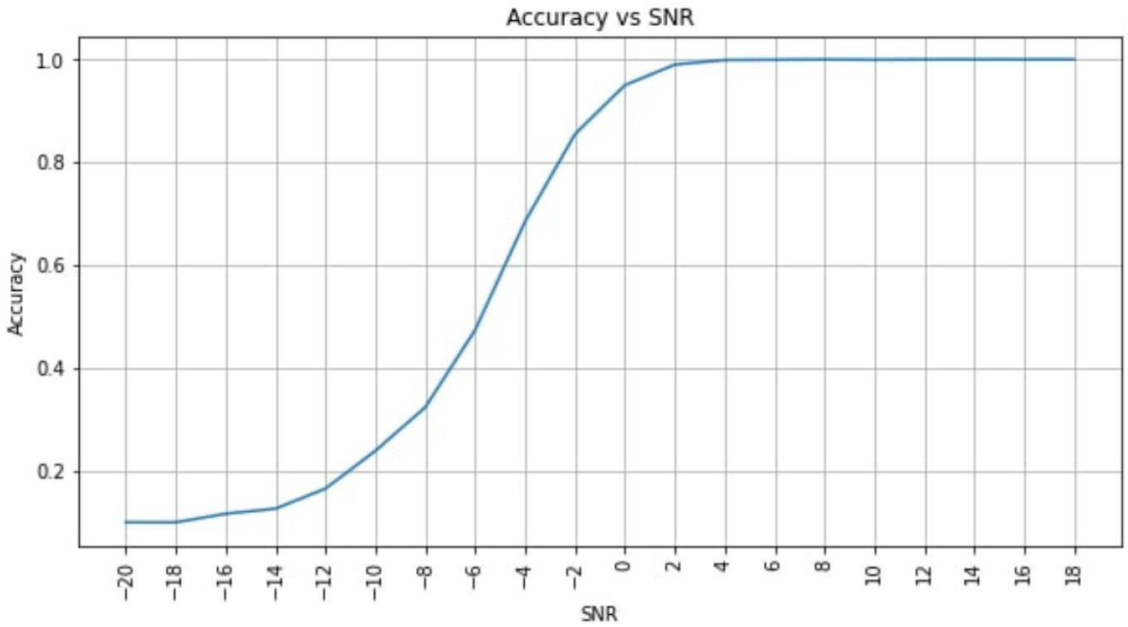


Figure 3- 28 Accuracy vs SNR for the jamming detection model 3 using distribution#5

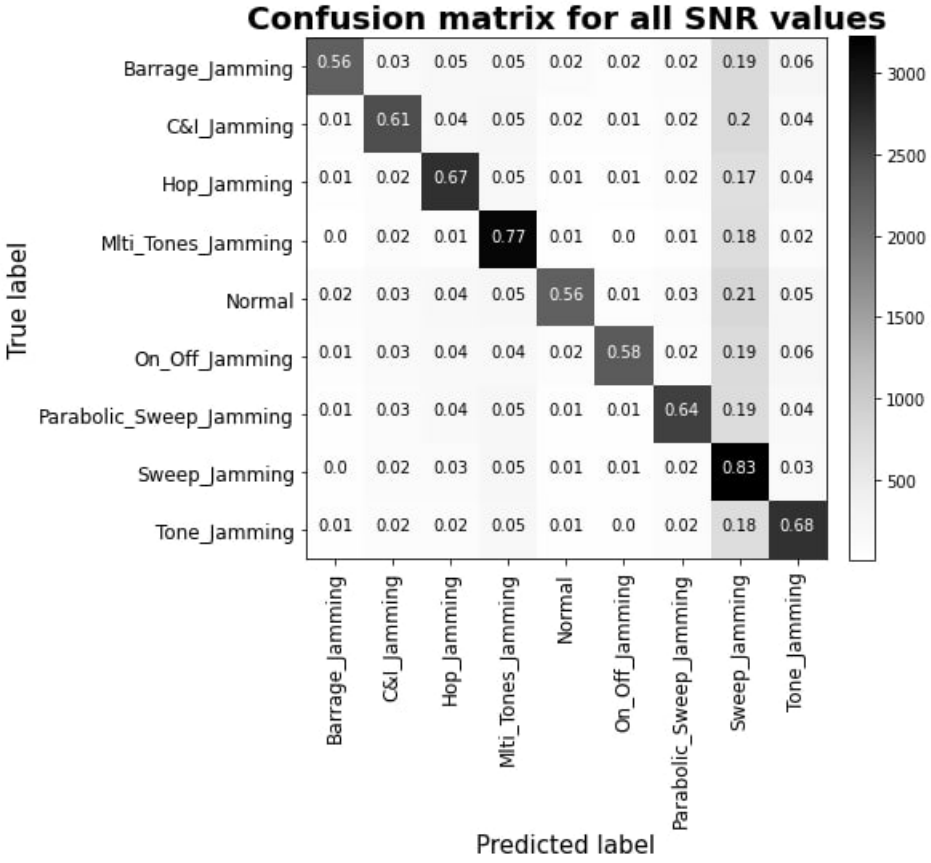


Figure 3- 29 Confusion matrix of Model3 using STFT and distribution #5 for all SNR levels

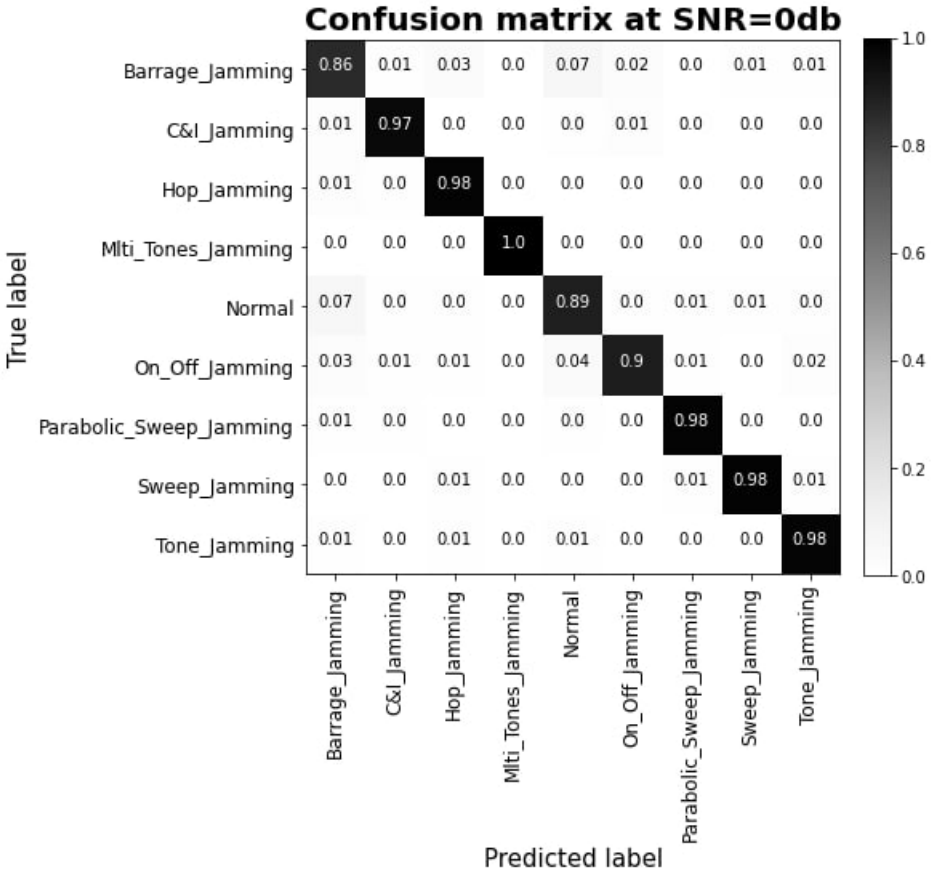


Figure 3- 30 Confusion matrix of Model3 using STFT and distribution #5 for SNR levels above 0 dB

3.3.5 Model 4:

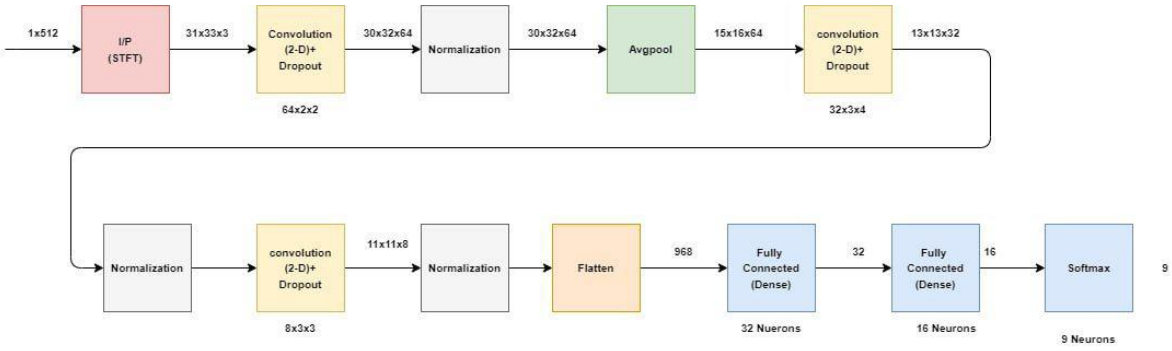


Figure 3- 31 Model 4 block diagram

We developed this model based on a dataset with only a QPSK modulation scheme, that was based on distribution#5 mentioned in section 2.3.1, and a quick view on this data we set the jamming power to 0 dB, and this is a relatively high value compared to the old datasets. The accuracy didn't increase, because we added more randomization across SNR levels, as now each frame has unique values of noise and jamming added to the signal, the average accuracy for this model was 64.35% over all SNR levels, and 97.52% for signals above zero dB, the total number of parameters for this model was 60,561. Figures 3-32, 3-33, 3-34 shows the result of the model.

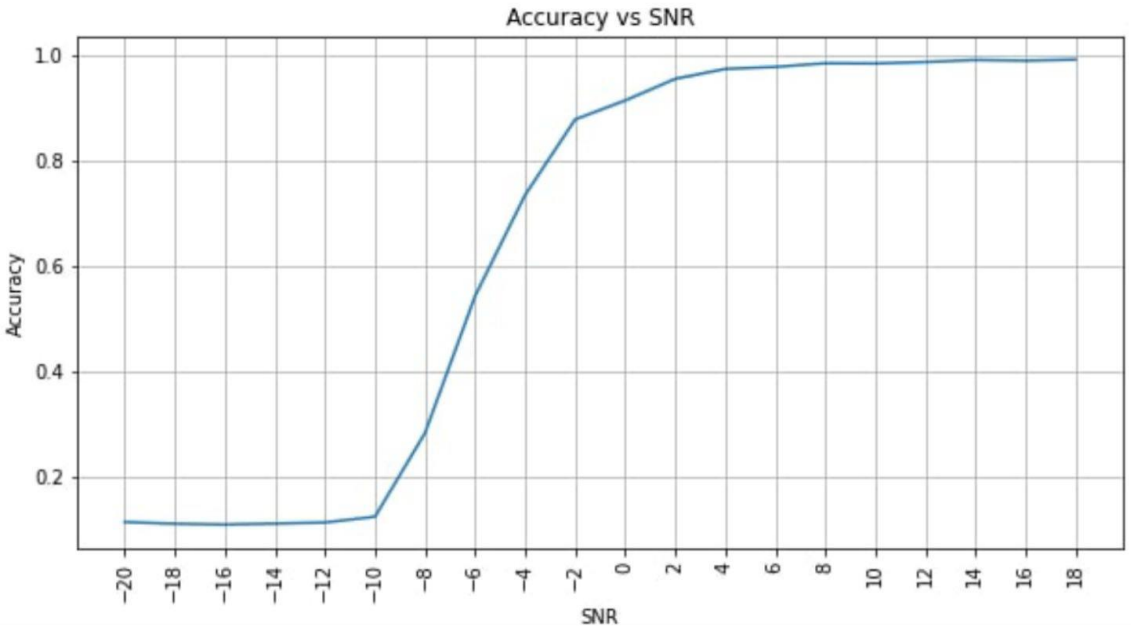


Figure 3- 32 Accuracy vs SNR for model 4 using only QPSK and jamming power 0 dB

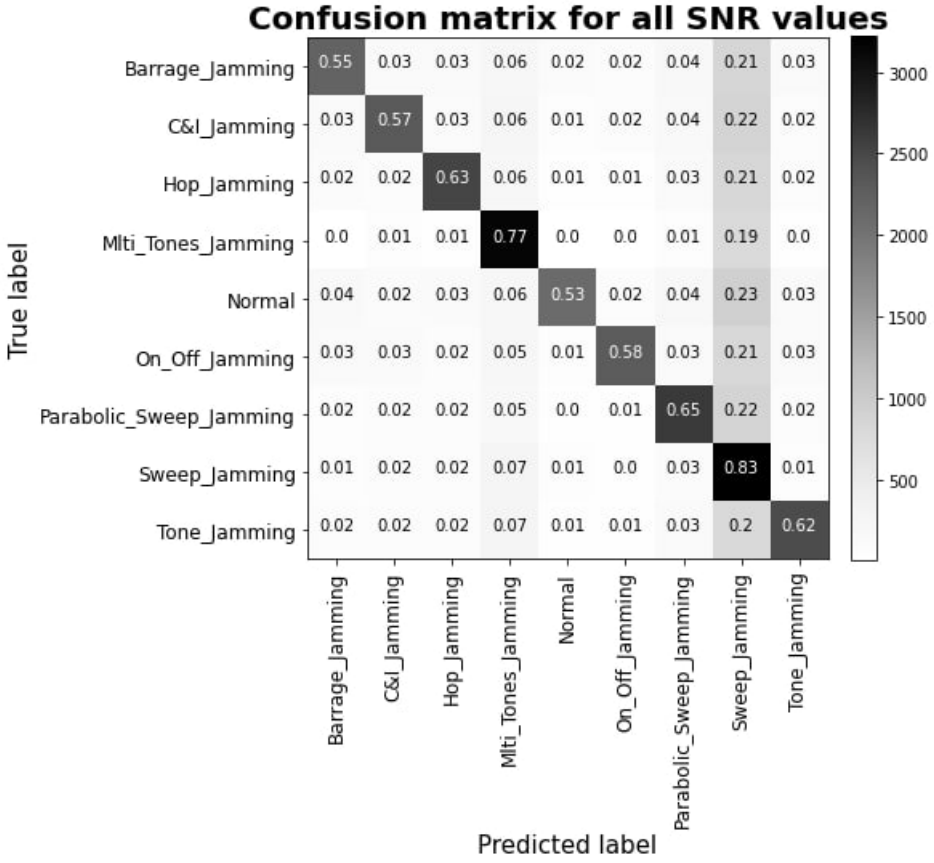


Figure 3- 33 Confusion matrix of Model4 using only QPSK distribution#5 for all SNR levels

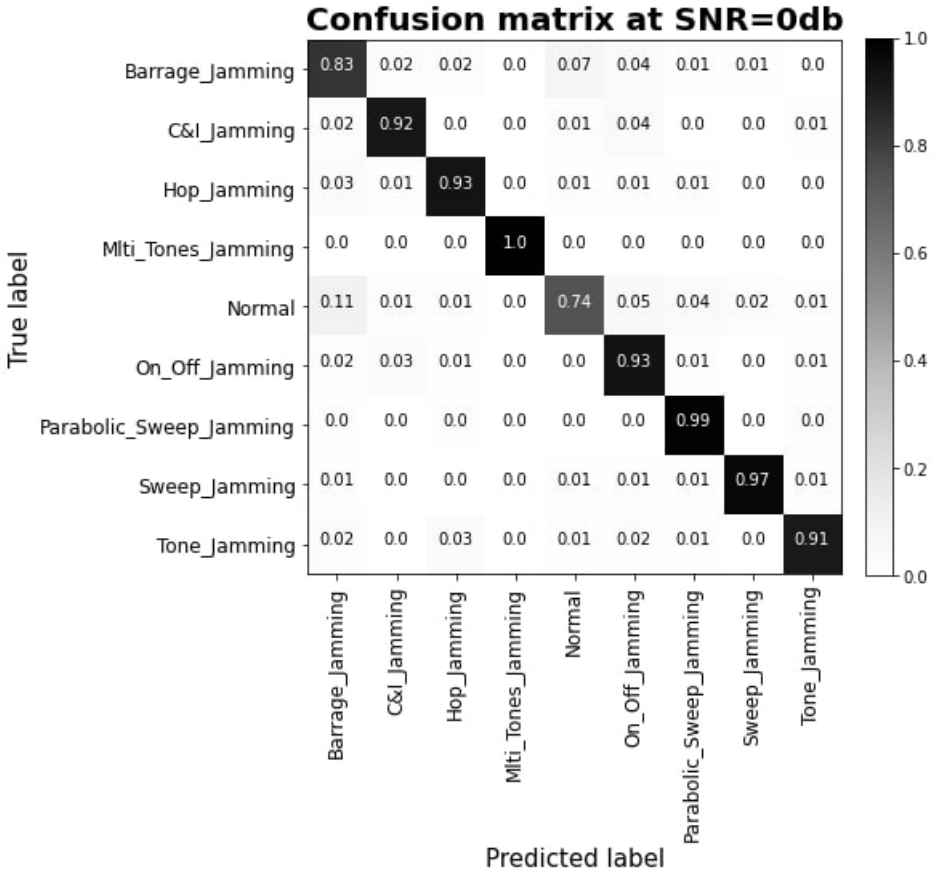


Figure 3- 34 Confusion matrix of Model4 using QPSK and distribution #5 for all SNR of 0 dB

3.3.6 Model 5:

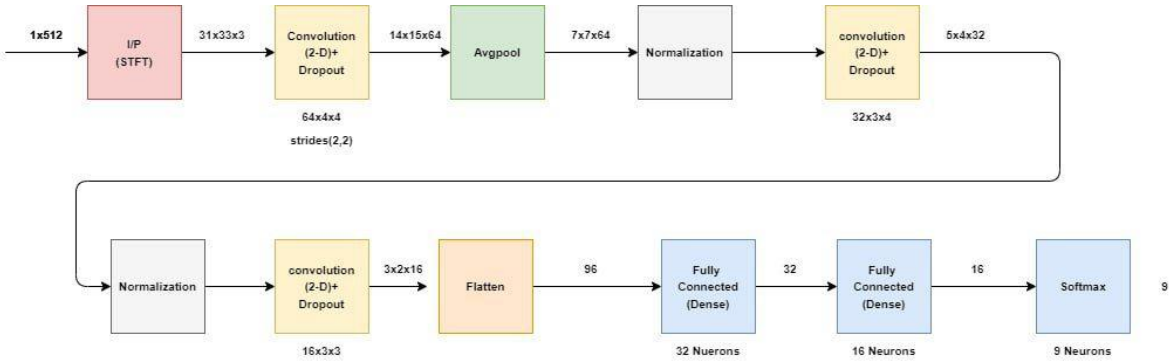


Figure 3- 35 Model 5 Block diagram

This is the final model on the GPU, this was developed with a dataset with QPSK modulation scheme only, and we set the jamming power from -4dB to 0dB, and we separate the jamming channel from the data channel, this separation makes it very hard for the model to detect the jamming, and the jamming power is less than the last dataset. After running the model on the dataset with the previous specifications, we obtained an average accuracy of 56.89% over all SNR level and 93.31% for SNR levels above zero dB, and the total number of parameters for this model was 36,537. Figures 3-36, 3-37, 3-38 shows the result of the model.

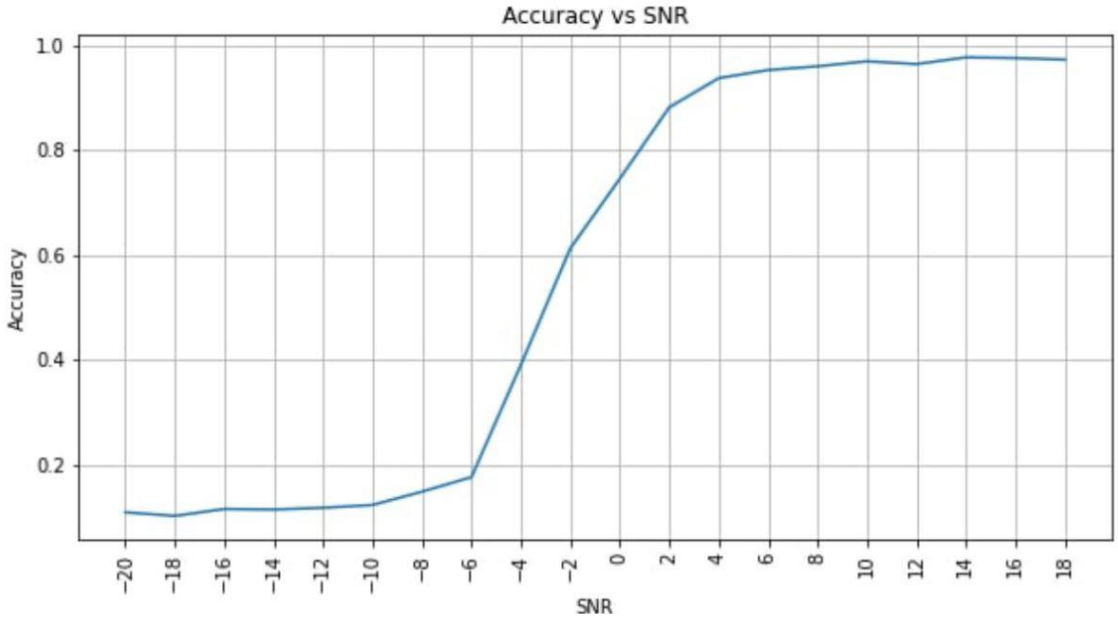


Figure 3- 36 Accuracy vs SNR for the final model

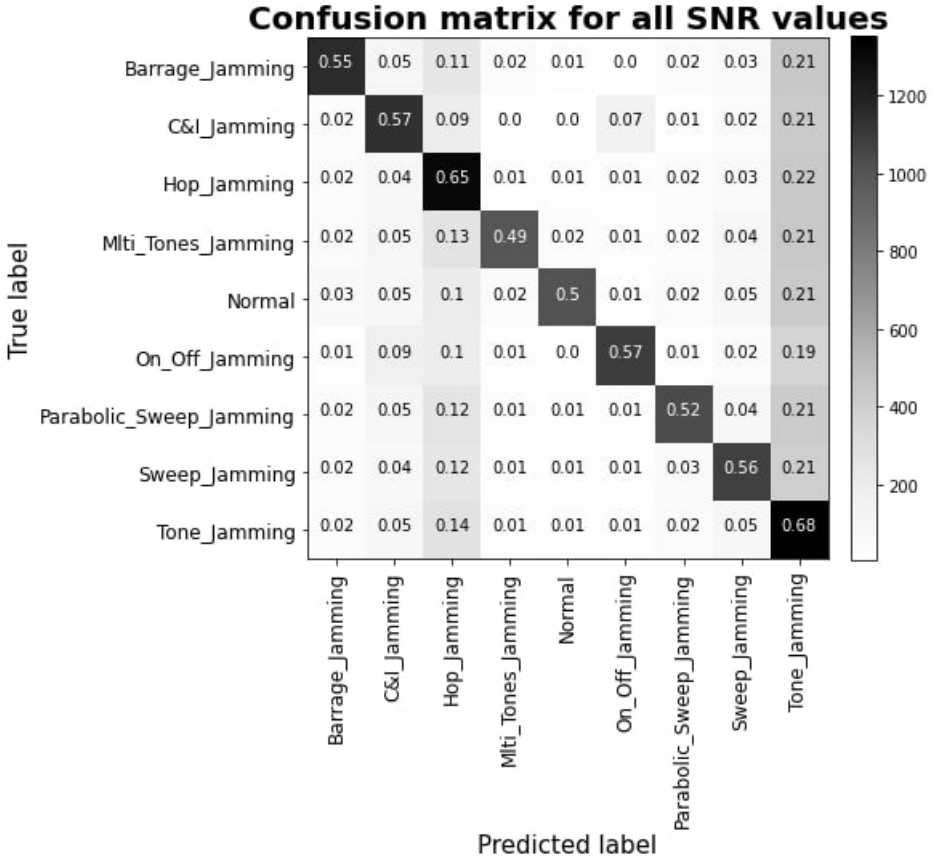


Figure 3- 37 Confusion matrix of Model5 for all SNR levels

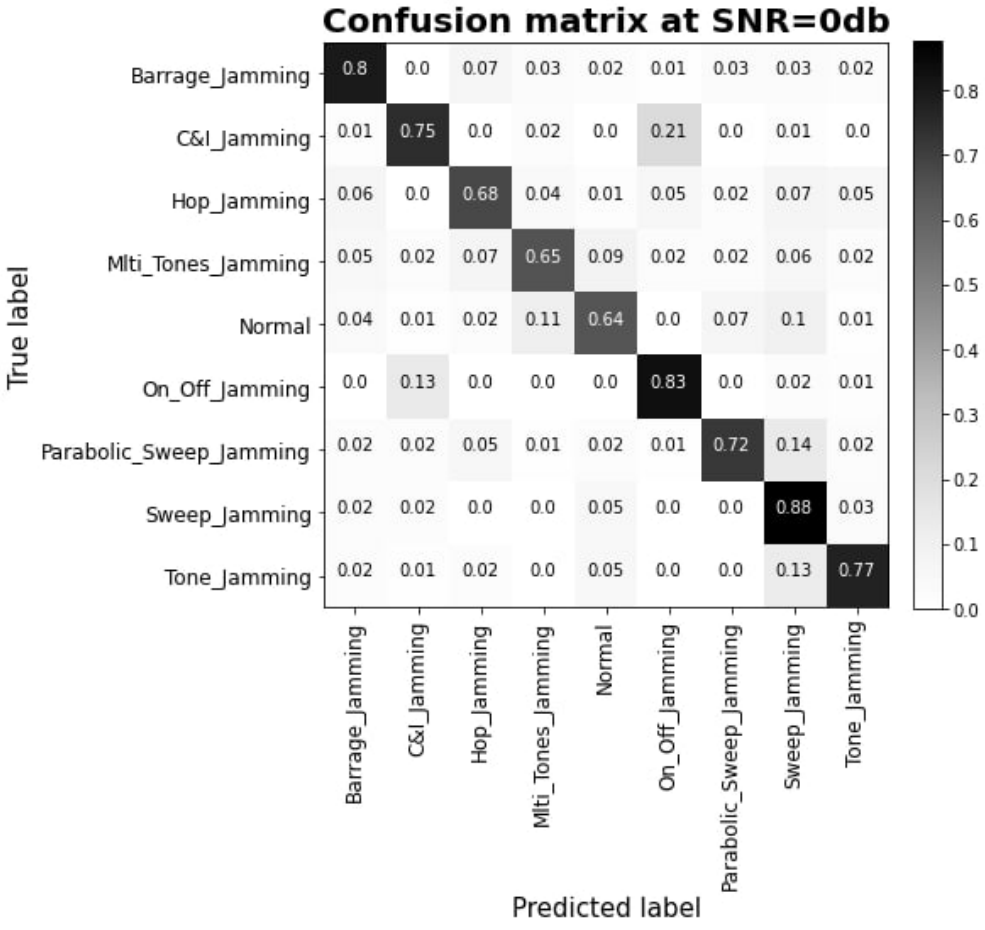


Figure 3- 38 Confusion matrix of Model5 for SNR level of 0 dB

We applied this model on a dataset that contains only jamming, because we thought this case might happen when we sense the spectrum to start our communication link, the results of this dataset was that the average accuracy is 58.83% over all JNR levels, and for jamming power above zero dB, we obtained an average accuracy of 96.08%. Figures 3-39, 3-40 shows the result of the model.

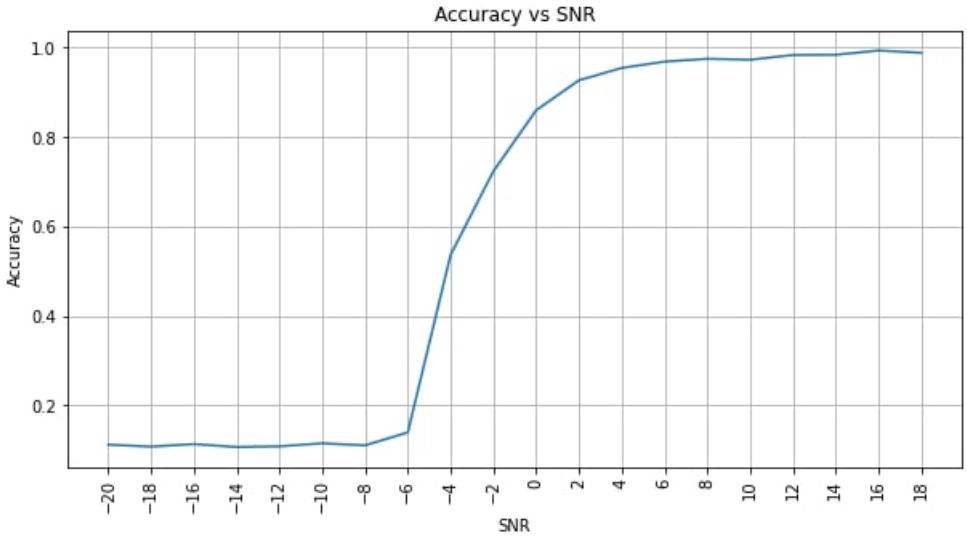


Figure 3- 39 Accuracy vs SNR for jamming only case using model5

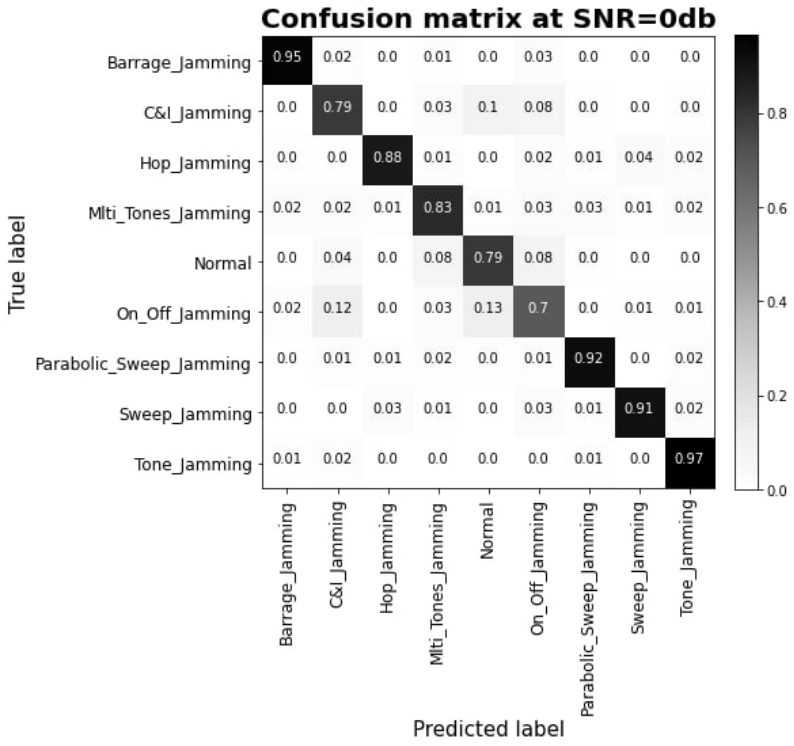


Figure 3- 40 Confusion matrix of jamming only dataset using Model 5

We used this model on a dataset that contains jamming with power equal to 0 dB, and the model was able to reach an average accuracy of 70.95% over all SNR levels, and an average accuracy of 96.54% for SNR levels above 0 dB. Figures 3-32, 3-33, 3-34 shows the result of the model.

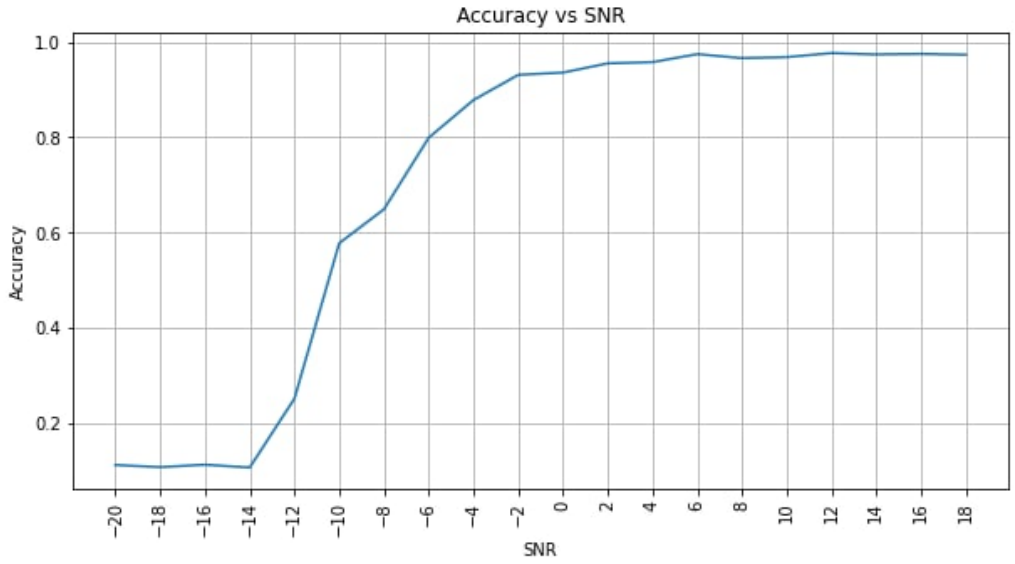


Figure 3- 41 Accuracy VS SNR for Model 5 and jamming power of 0 dB

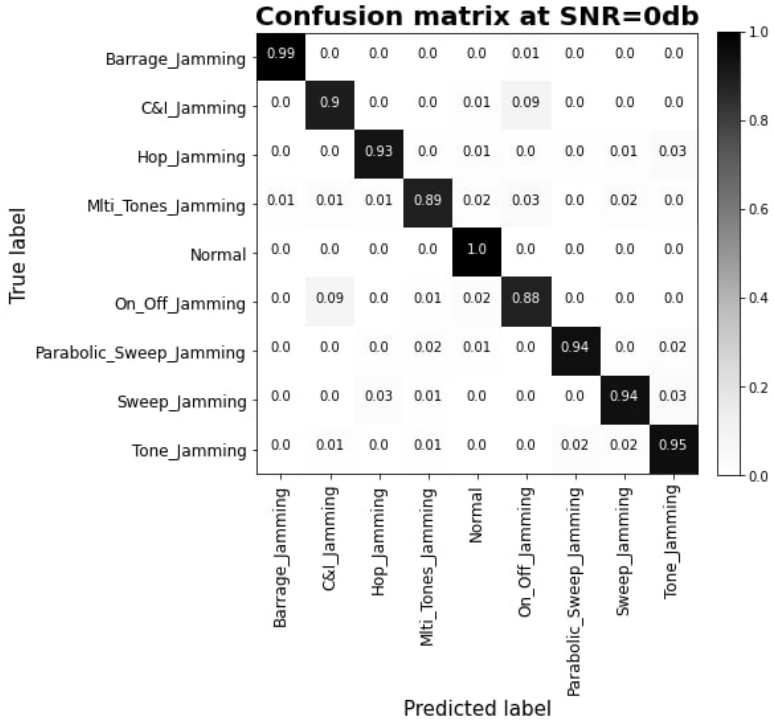


Figure 3- 42 Confusion matrix of Model 5 using jamming power of 0 dB.

Due to the variation of accuracy when changing the jamming power we wanted to test the range in which the model will behave correctly for changing the jamming power, so we used this model with another dataset, this dataset has 11 steps of different jamming power levels varying from -10 dB to 10 dB with a step of 2 dB. In each level of jamming power, we have 20 levels of SNR, form -20 dB to 18 dB that has a step of 2dB. We generated this data set to know the range of jamming that the model can still detect the jamming. The dataset consisted of 110,000 frame, all of them were modulated by QPSK only, and for each level of the 11 levels of jamming power we had 10,000 frame distributed between 20 SNR levels, as we have 500 frames for each SNR level, and this is relatively a small number, but we have limited resources, and this was the maximum number of data frames that the memory can

handle them. In this dataset, we used 10% of the data as a test data, or in other words, 50 frames for each SNR level and a certain jamming power, and 10% for validation, and 80% for training, and as we said, the dataset was small, this leads somehow to a zigzag shape in the accuracy Vs. SNR graph, and we obtained the 3-D graph in figure 3-43:

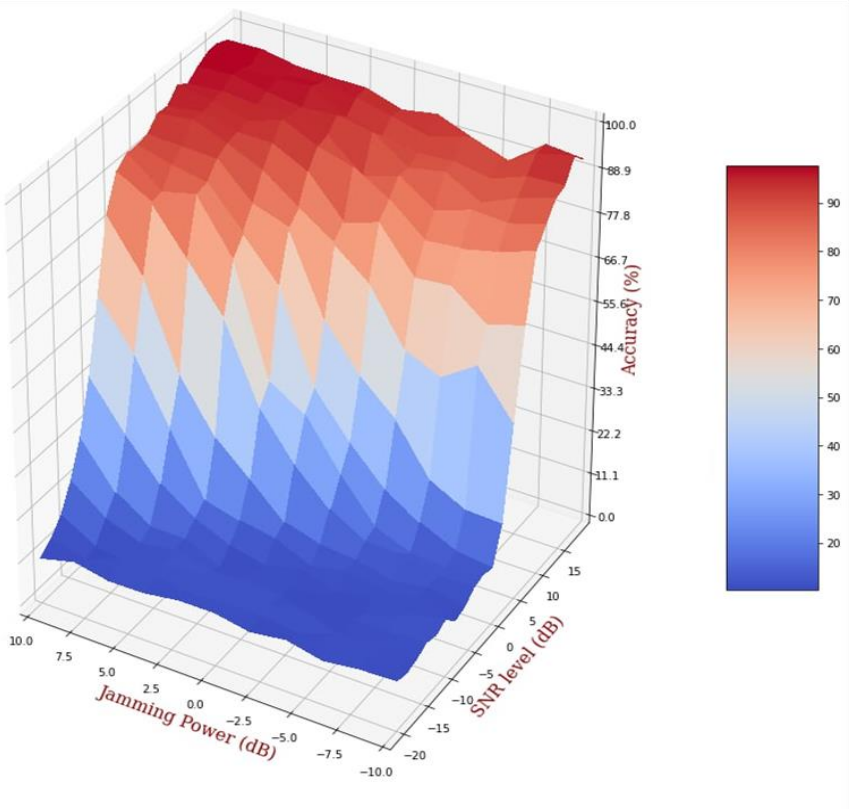


Figure 3- 43 Accuracy Vs SNR Vs Jamming Power

The average accuracy for each jamming power is as follows:

Table 3- 1 Model Average accuracy VS changing jamming power

Jamming Power	Average Accuracy %		Jamming Power	Average Accuracy %
-10 dB	39.96		2 dB	53.46
-8 dB	40.00		4 dB	58.66
-6 dB	39.49		6 dB	62.43
-4 dB	41.34		8 dB	66.47
-2 dB	44.96		10 dB	70.27
0 dB	48.36			

As shown in table 3-1 as expected, increasing the jamming power makes it easier for the model to detect the jamming type.

These two graphs are for accuracy vs. snr at -10dB and 10db to show the two extreme cases:

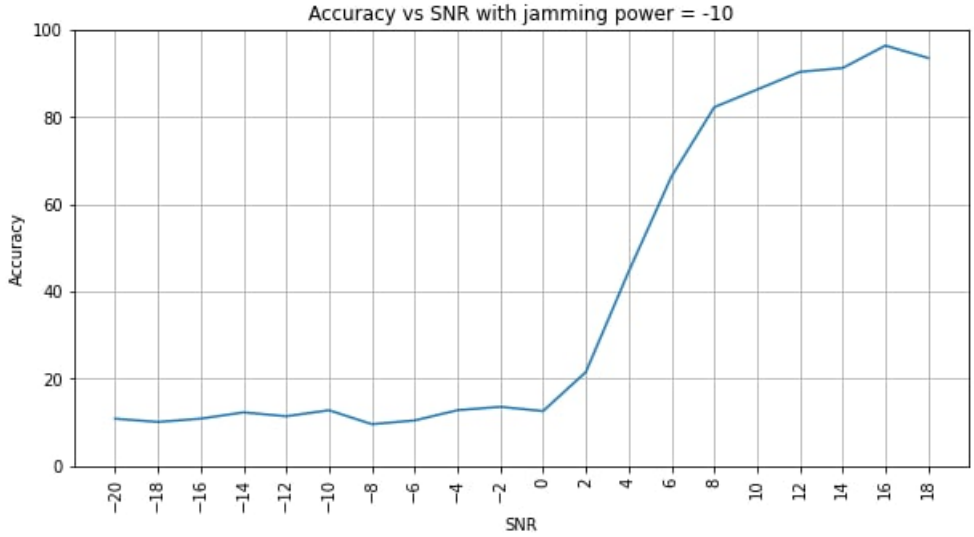


Figure 3- 44 Accuracy Vs SNR with jamming power = -10 dB

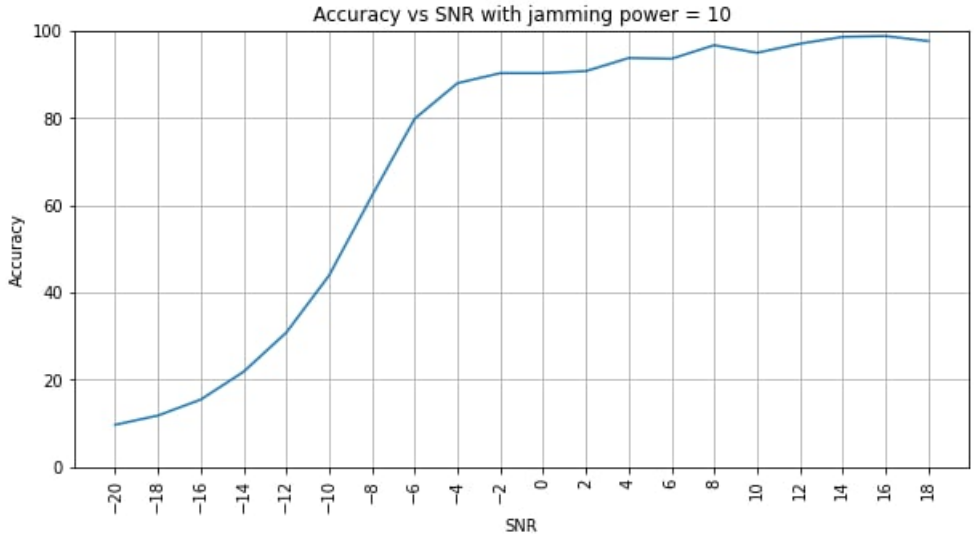


Figure 3- 45 Accuracy Vs SNR with jamming power = 10 dB

3.4 Model development for Digital implementation

3.4.1 GPU model modifications:

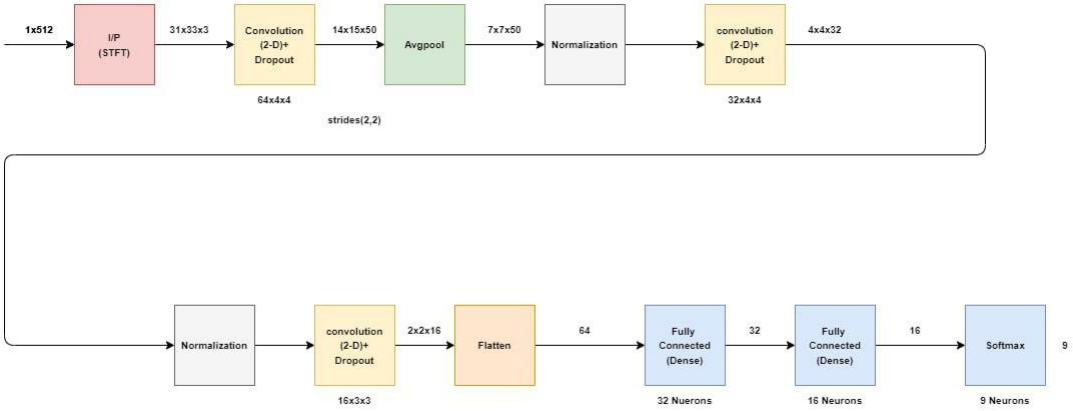


Figure 3- 46 Modified Model block diagram

In order to make it simple in the digital design and implementation part we used model5 and made some simplification on it. This simplification was made in two parts. First, the second convolutional layer's filter shape became 4x4 instead of 3x4 in the original model, this shape is like the first filter shape, which makes it easier in digital implementation. Also in order to reduce the number of parameters and the number of neurons in the flatten layer, we reduced the number of filters in the first convolutional layer from 64 filters to 50 filters, which is the second part of simplification, this makes the flatten layer contain only 64 neuron instead of 96 neuron. The total number of parameters for this model are 35,467 parameters. The output of the STFT is a complex number, so we take the real part in the first channel, then the imaginary part in the second channel, and the squared magnitude in the third channel to avoid designing the square root in digital design, because it is relatively a hard problem in the digital part.

The obtained result from this model we could reach an average accuracy of 68.99% over all SNR levels instead of to 70.95% in the original model,

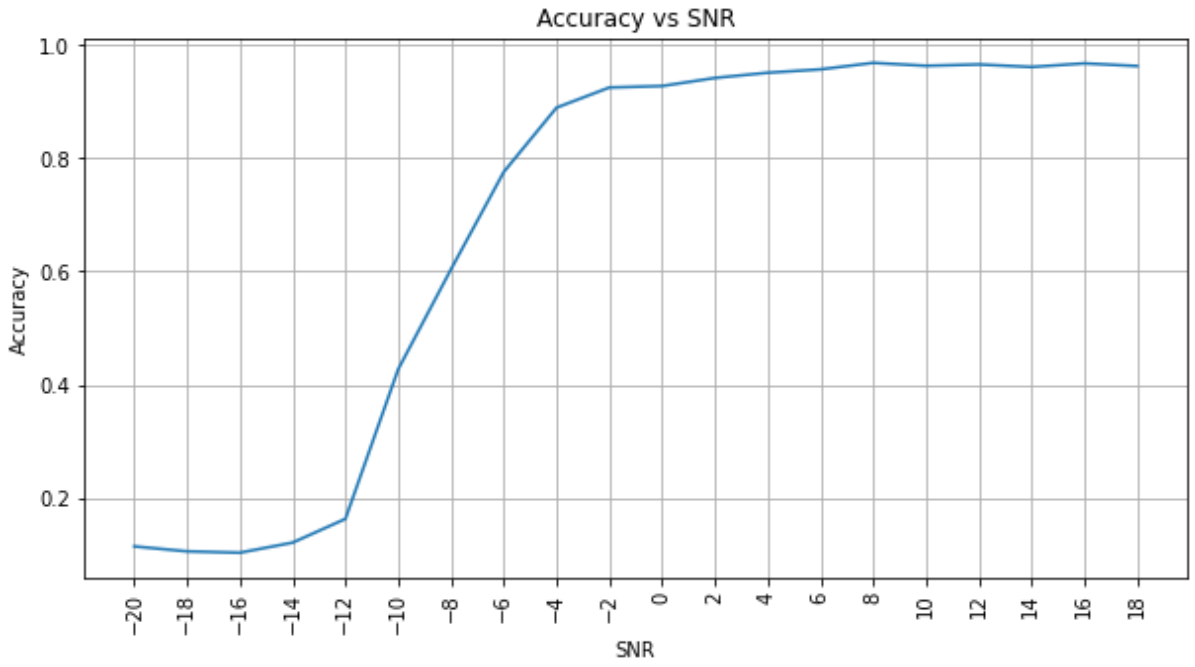


Figure 3- 47 Simplified model Accuracy Vs. SNR

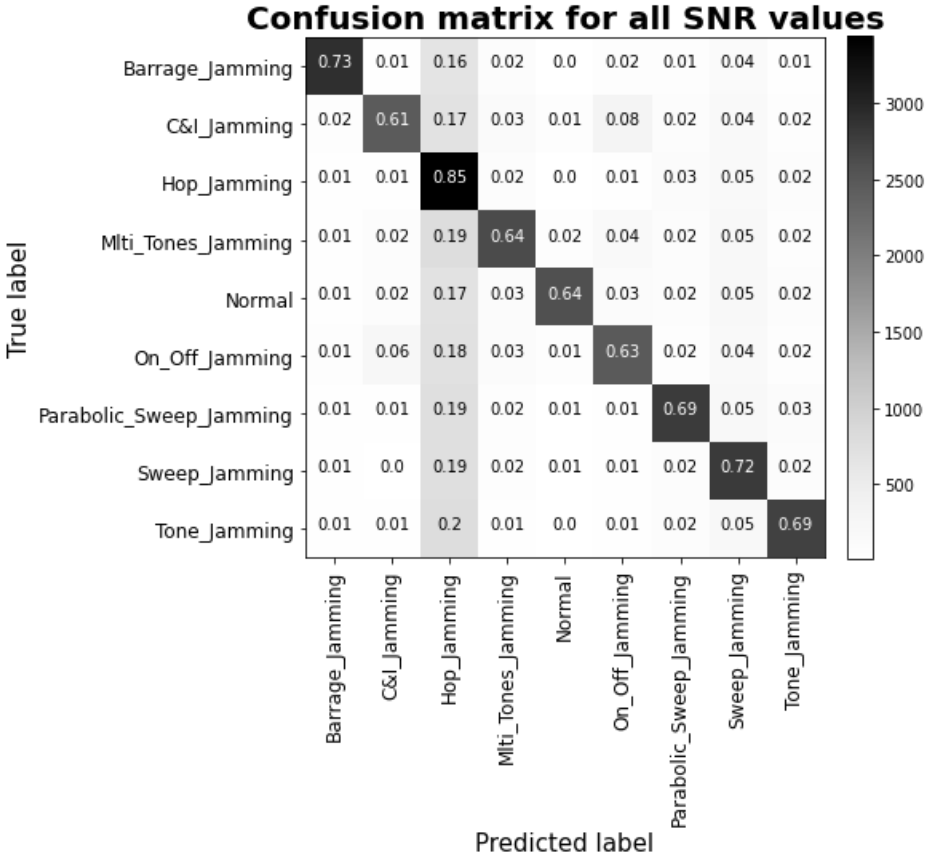


Figure 3- 48 Confusion matrix of the simplified model for all SNR levels

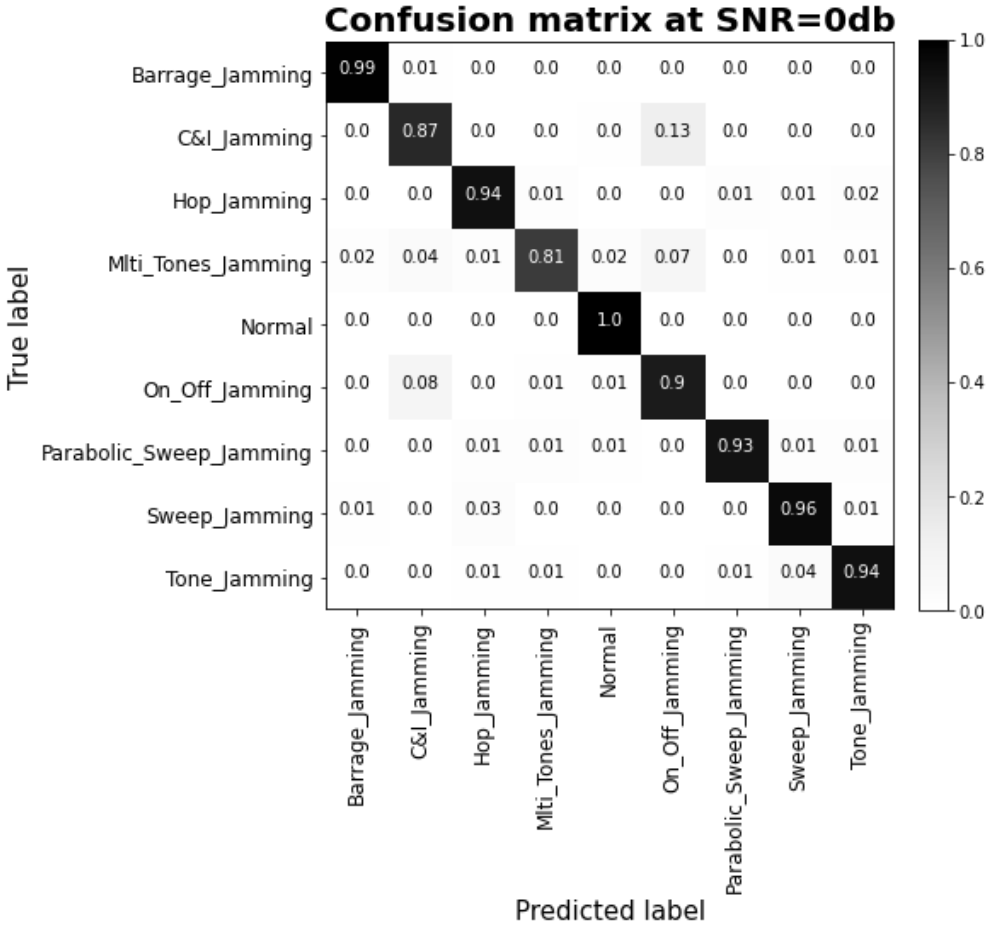


Figure 3- 49 Confusion matrix of the simplified model for SNR of 0 dB

3.4.2 Model simplification:

Due to the nature of the STFT that has a high peak to average ratio, using the square of the magnitude will be even larger, and the largest number observed in the model is 8467.486328125 which requires 14 bits for decimal part and 9 bits for the fraction part, and 1 bit for the sign to be implemented in digital design, this leads to a word size of 24 bits. After the first convolution layer, the maximum number observed was 21.57641792, which will need only 5 bits for the decimal part, so we decided to normalize the input of the STFT, which makes it easy to take a smaller number for the decimal part, but using normalization made the simplified model fail on the GPU. Therefore, we tried to use another approach, which rely on discarding the large numbers obtained from the STFT. This approach due to the following reasons:

1- When we investigate the output of the STFT we notice the following
   a. The histogram of the output shows that large output numbers occur much less frequent than the smaller output numbers as shown in figure 3-50
   b. Most of the output numbers are in range from 0 to 30, hence they can be represented in 5 bits only for the integer part, which gives an accuracy up to 32
2- CNN has high immunity to data changes
3- DSP slices on FPGA can handle up to 22 bits multiplication

For the above reasons we can deduce the following:

1- We can assume all numbers above 32 as outliers and discard them.
2- The word size can be 22 bits; 1 for the sign, 6 bit for integer part, even though 5 bits is enough which allows for some safety factor, and 15 for the fraction part.

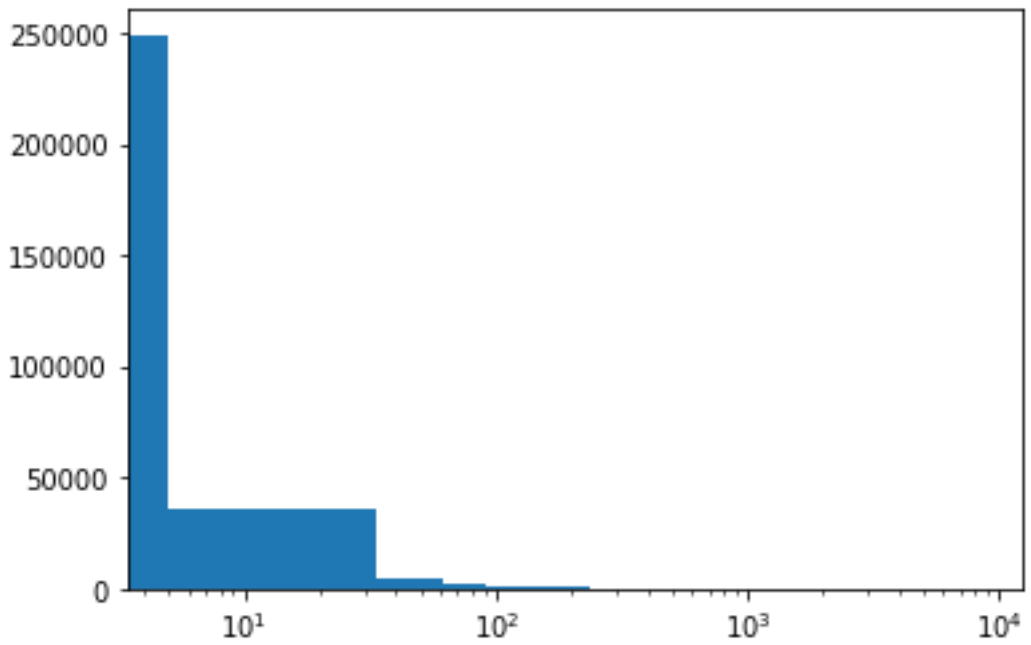


Figure 3- 50 Histogram of the STFT output values

Another way used to simplify the model to be easier in digital implementation was merging the batch normalization layer with the previous convolutional layer, to make this, we

didn't set the activation function in the convolutional layer, instead we apply the ReLU activation function after the batch Normalization layer, the mathematics behind this is as follows:

$$y_1 = wx + b \rightarrow (1)$$

Eq. no.(1) is general equation of fully connected layer, but we can apply the same logic in convolution layer.

$$y_2 = \frac{\gamma(x_i - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \rightarrow (2)$$

Eq. no.(2) is the equation of batch normalization, it was mentioned before in section 3.1.3.10, and since we don't have activation function between the two layers, we can substitute the input to the batch normalization layer from the output of the first layer that in Eq. no.(1) in Eq. no(2), and this will lead to Eq. no.(3)

$$y_2 = \frac{\gamma(wx + b - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \rightarrow (3)$$

In equation (4), we reached the last form of the equation that will be used in replacing the weights of the convolutional layer.

$$y_2 = \frac{\gamma w}{\sqrt{\sigma^2 + \epsilon}} x + \frac{\gamma(b - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \rightarrow (4)$$

As we will replace the old weights (w) with $\frac{\gamma w}{\sqrt{\sigma^2 + \epsilon}}$, and the free term 'bias (b)' with $\frac{\gamma(b - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta$.

Figure 3-51 shows the weights of the model in the normal operation

Weights:

Name	Conv_1	Conv_2	Conv_3
shape	(4, 4, 3, 64)	(4, 4, 64, 32)	(3, 3, 32, 16)
max_value	0.4756105	0.26042637	0.5539698
min_value	-0.52537346	-0.23878282	-0.5395579
abs_min_value	3.8603568e-05	4.0367486e-06	2.6594014e-05

Biases:

Name	Conv_1	Conv_2	Conv_3
shape	(64,)	(32,)	(16,)
max_value	0.007040187	0.043417472	0.12243499
min_value	-0.0020475981	-0.05869098	-0.024891092
abs_min_value	6.149016e-07	0.00027933565	0.0062795416

Figure 3- 51 Weights and biases of the Convolution layers before removing batch normalization

After applying the previous transformations, the weights of the model becomes:

Weights:

Name	Conv_1	Conv_2	Conv_3
shape	(4, 4, 3, 50)	(4, 4, 50, 32)	(3, 3, 32, 16)
max_value	0.6502432	0.28422812	0.5457495
min_value	-0.5773837	-0.24685806	-0.6845364
abs_min_value	3.8763592e-05	2.5180125e-06	5.734693e-05

Biases:

Name	Conv_1	Conv_2	Conv_3
shape	(50,)	(32,)	(16,)
max_value	0.0042525497	0.09046292	0.08473538
min_value	-0.0066546835	-0.1407807	-0.051909722
abs_min_value	2.6567471e-07	0.0035532357	0.00569941

Figure 3- 52 Weights and biases of the Convolution layers after removing batch normalization

Weights:

Name	Fully_connected_1	Fully_connected_2	Fully_connected_3
shape	(64, 32)	(32, 16)	(16, 9)
max_value	0.9273292	0.88130814	0.9565967
min_value	-1.1378609	-0.9141545	-1.0096571
abs_min_value	0.00015869156	0.0010227248	0.010686959

Biases:

Name	Fully_connected_1	Fully_connected_2	Fully_connected_3
shape	(32,)	(16,)	(9,)
max_value	0.12997985	0.15085855	0.07988813
min_value	-0.070890196	-0.21230653	-0.0851886
abs_min_value	0.00090439385	0.004223055	0.00490177

Figure 3- 53 Weights and biases of the fully connected layers after removing batch normalization

Chapter 4: Digital Design Part

4.1 Methods to improve the efficiency of deep learning implementation

In this section, we are going to talk about some methods to improve the efficiency of deep learning implementation.

4.1.1 Pruning

Pruning is defined as discarding less important neuron without changing the original network structure as shown in Figure 4-1, to make the network size smaller and to alleviate over-fitting, without affecting the accuracy of original network.

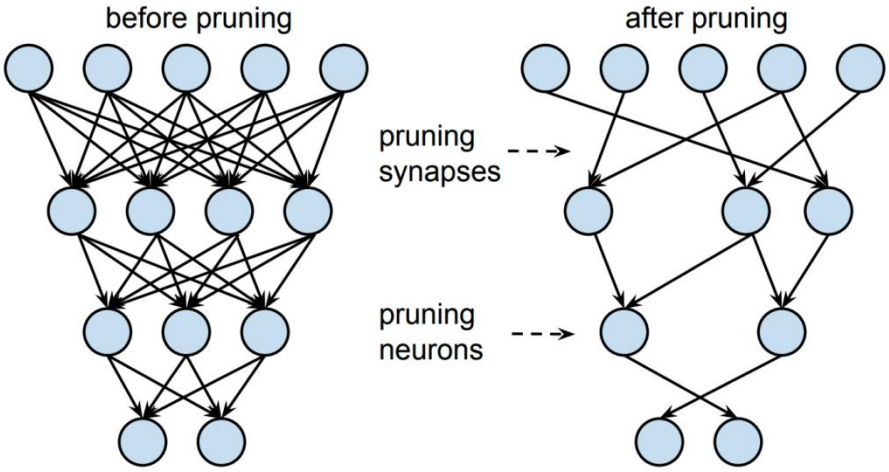


Figure 4- 1 effect of pruning in neural networks

Pruning method has three steps; the first step is learning the connectivity via normal network training to learn which connections are important. Unlike conventional training that used to learn the final values of the weights. The second step is to prune all connections with weights below a threshold are removed from the network. The final step is retraining the network to learn the final weights for the remaining sparse connections.

Pruning is an effective way of reducing the area used in the FPGA in a trade of with small part of the model accuracy but since we have a relatively small architecture of the model and we are more interested in having the best possible accuracy we did not use this method in our model

The difference between dropout and pruning is that in dropout, each parameter is probabilistically dropped during training, but will come back during inference. In pruning, parameters are dropped forever after pruning and have no chance to come back during both training and inference

4.1.2 Quantization

Network quantization and weight sharing compresses the pruned network by reducing the number of bits required to represent each weight. In the context of deep learning, the predominant numerical format used for research and for deployment has so far been 32-bit floating point. However, the desire for reduced bandwidth of deep learning models has driven research into using lower-precision numerical formats. It has been extensively demonstrated that weights and activations can be represented using 8-bits without getting significant loss in

accuracy for deep learning models that work with images but since our model take it's input after STFT pre-processing which have the problem of high peak to average ratio, the quantization with only 8 bits in our model will degrade the accuracy of the model significantly so for our model we used a higher number of 22 bits.

4.1.3 Late down sampling

Each convolution layer in a convolutional network produces an output activation map with a spatial resolution that is at least 1x1 and often much larger than 1x1. The height and width of these activation maps are controlled by: the input data size and the down sample layers in the architecture. Down sampling is often engineered into CNN architectures by pooling layers by setting the (stride > 1) in some of the convolution.

If most layers in the network have a stride of 1, and the strides greater than 1 are moved towards the end of the network, then large activation maps will be in many layers in the network.

This method is not applied in our case since the data distribution in our dataset is coming from STFT and we are usually detecting patterns in those peaks and since the using of smaller filters gives more local information and using large filters will give more global information so we decided to use stride higher than 1 and filter size relatively large in the beginning of the network which gives higher detection accuracy to our model.

4.1.4 DSD: Dense-Sparse-Dense Training

DSD produces same model architecture but can find better optimization performance by regularizing deep neural networks. The first step "Dense" is training a dense network to learn important weights. The second step "Sparse" is pruning the network and retraining the network to learn the final weights for the remaining sparse connections. The final step "re-Dense" is increasing the model capacity by reinitializing the pruned parameters from zero and retrain the whole dense network as shown in Figure 4-2.

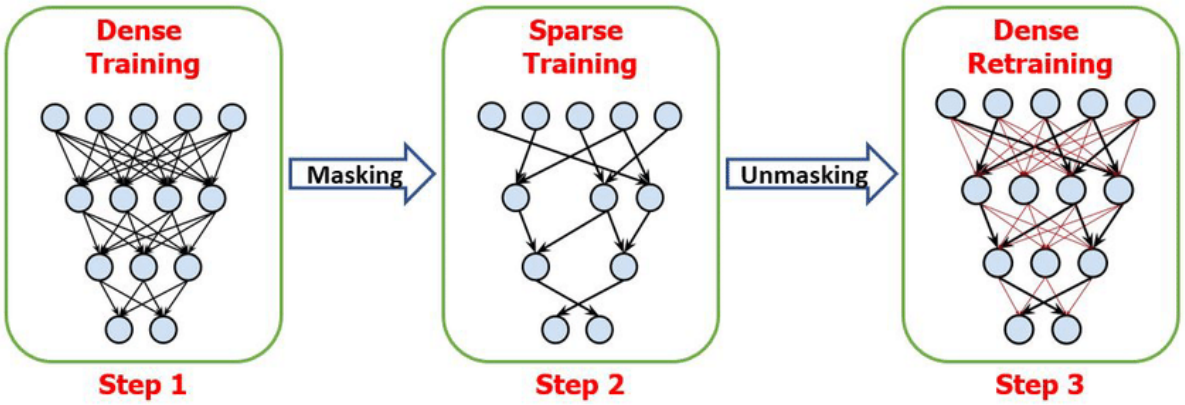


Figure 4- 2 DSD steps

DSD and Dropout both are used to prevent over-fitting and regularize the network. But the difference is that DSD training learns with a deterministic data driven sparsity pattern but Dropout uses a random sparsity pattern at each SGD iteration.

4.1.5 Parallelism

Parallelism means many calculations or the execution of processes are carried out simultaneously, that will decrease the time of execution of CNN architectures. For example, two-dimensional convolution is computed between sliding windows of input feature maps and kernels, and consumes most computation time of CNN as shown in Figure 4-3. Parallelism included inside an output feature map of each layer, known as intra-output parallelism.

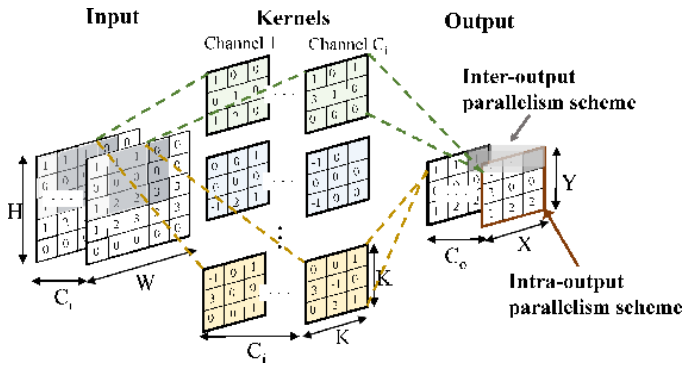


Figure 4- 3 parallelism in CNN network

4.1.6 Pipelining

The approach is to rearrange the algorithm into a pipeline, where each stage can operate simultaneously with the other stages as shown in Figure 4-4 Pipelining tends to be faster and it can even be more resource efficient

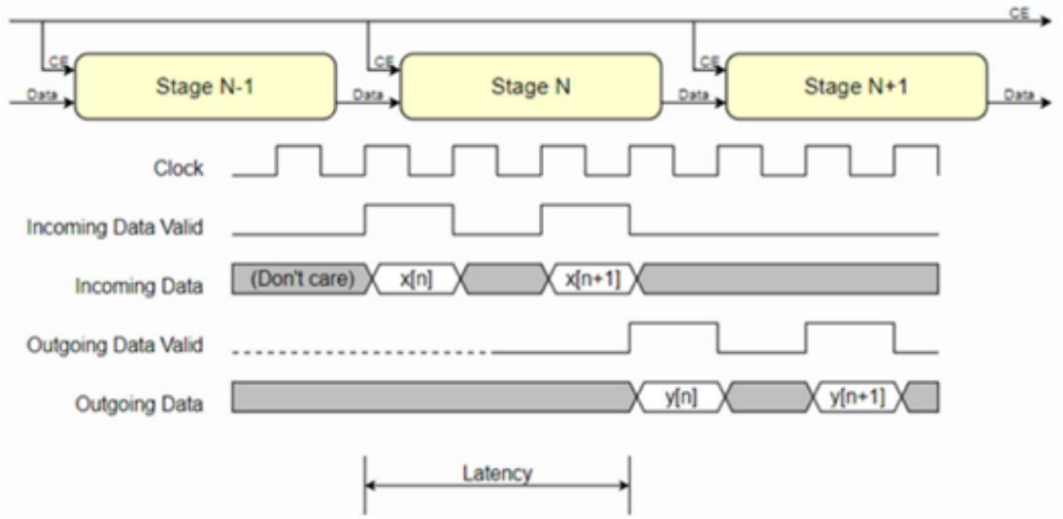


Figure 4- 4 Pipelining

4.2 FPGA

4.2.1 Introduction

An FPGA (Field Programmable Gate Array) is an IC consisting of programmable logic gates and interconnections that can be programmed using an HDL (hardware descriptive language) to do a specific function. It is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), for example VHDL or Verilog.

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR gates. Also, FPGAs contain memory elements, which may be simple flip-flops or more complete blocks of memory. An FPGA can be used to solve any problem which is computable. This is trivially proven by the fact that an FPGA can be used to implement a soft microprocessor.

Most of the digital applications can be implemented with powerful specialized processors, but FPGAs are sometimes significantly faster for some applications because of their parallel nature and optimality in terms of the number of gates used for a certain process. In all microprocessor-based systems, the functions are executed sequentially, one line of code after another. On the other hand, FPGAs execute their operations in parallel, so FPGAs can be much faster in many applications where speed is crucial. FPGAs also offer great flexibility; the same FPGA IC can be used as a missile guiding system or just a network processing device. As their size, capabilities, and speed increased, they took over additional functions to the point where some are now marketed as full systems on chips (SoC). Particularly with the introduction of dedicated multipliers into FPGA, applications which had traditionally been the sole reserve of DSPs (Digital Signal Processors, a specialized type of processors for signal processing) began to use FPGAs instead.

Another trend in the use of FPGAs is hardware acceleration, where one can use the FPGA to accelerate certain parts of an algorithm that need parallel processing and share part of the computation between the FPGA and a generic processor.

4.2.2 FPGA Internal Components

Obviously as shown in Figure 4-5, FPGAs do not only consist of logic gates or look-up tables only. They are made up of many types of logic blocks for implementing many functions and to increase its flexibility.

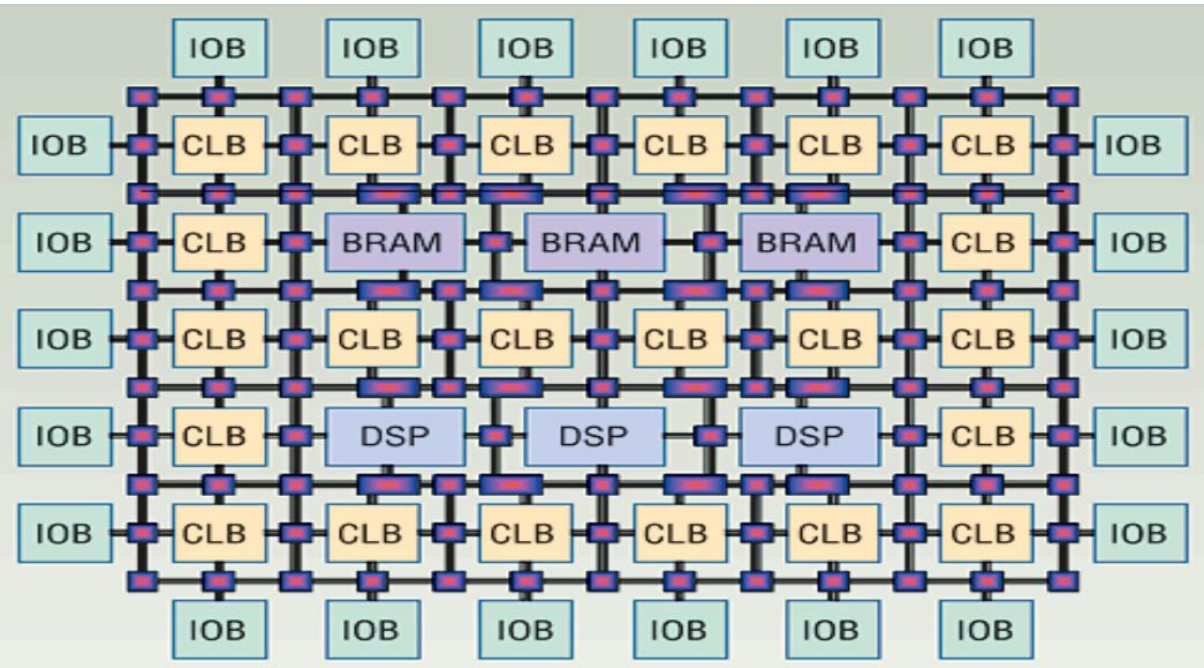


Figure 4- 5 FPGA internal Structure

4.2.3 Configurable Logic Blocks (CLBs)

These blocks contain the logic for the FPGA. In the dense architecture used by all FPGA vendors today, these CLBs contain enough logic to create a small state machine. The block contains ROMs for creating arbitrary combinatorial logic functions, also known as lookup tables (LUTs). It also contains flip-flops for clocked storage elements, along with multiplexers in order to route the logic within the block and to and from external resources. The multiplexers also allow polarity selection and reset and clear input selection as shown in Figure 4-6.

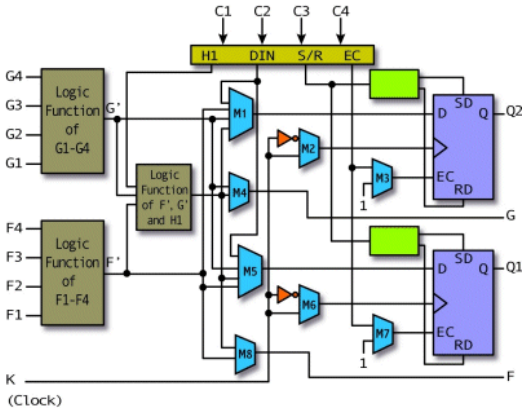


Figure 4- 6 Configuration logic block internal block diagram

4.2.4 Configurable I/O Blocks

A Configurable input/output (I/O) Block, is used to bring signals onto the chip and send them back off again. It consists of an input buffer and an output buffer with three-state and open collector output controls as shown in Figure 4-7. Typically, there are pull up resistors on the outputs and sometimes pull down resistors that can be used to terminate signals and buses without requiring discrete resistors external to the chip. The polarity of the output can usually be programmed for active high or active low output.

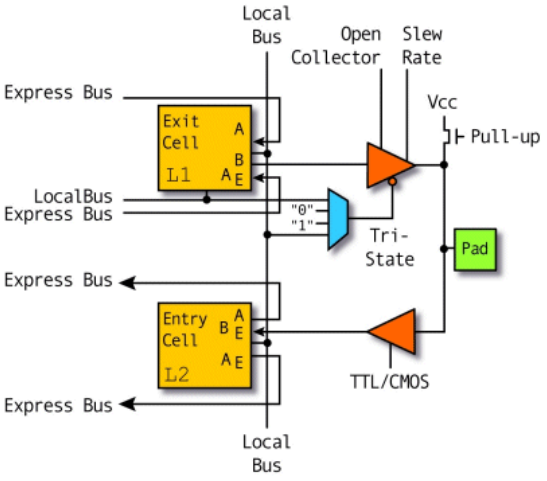


Figure 4- 7 Configuration input output internal block diagram

4.2.5 Programmable Interconnect

They are the long lines that can be used to connect CLBs to each other on the chip. These lines can also be used as buses within the chip as shown in Figure 4-8. Transistors are used to turn on or off connections between different lines. There are also several programmable switch matrices in the FPGA to connect the long and short lines together in specific, flexible combinations. Special long lines, called global clock lines, are specially designed for low impedance and thus fast propagation times. These are connected to the clock buffers and to each clocked element in each CLB. This is how the clocks are distributed throughout the FPGA, ensuring minimal skew between clock signals arriving at different flip-flops within the chip. In an ASIC, the majority of the delay comes from the logic in the design, because logic is connected with metal lines that exhibit little delay. In an FGPA, however, most of the delay in the chip comes from the interconnection, because the interconnection – like the logic – is fixed on the chip. In order to connect one CLB to another CLB in a different part of the chip often requires a connection through many transistors and switch matrices, each of which introduces extra delay.

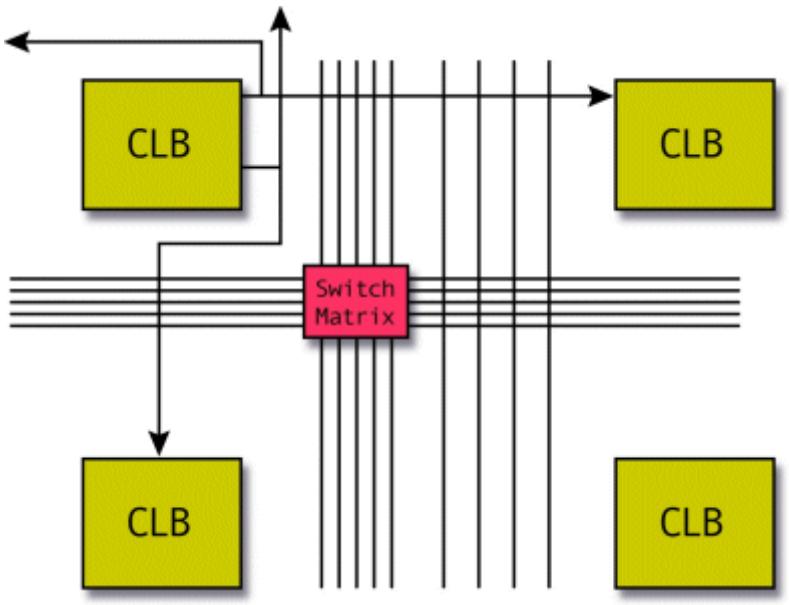


Figure 4- 8 Interconnection

4.2.6 Clock Circuitry

Special I/O blocks with special high drive clock buffers, known as clock drivers, are distributed around the chip. These buffers connect to clock input pads and drive the clock signals into the global clock lines described above. These clock lines are designed for low skew times and fast propagation times. Note that synchronous design is a must with FPGAs, since absolute skew and delay cannot be guaranteed anywhere but on the global clock lines.

4.2.7 Block RAM

It is a dedicated RAM block that stores data on the FPGA without consuming any additional LUTs in the design whereas distributed Ram is built up with LUTs. In terms of speed the distributed RAM is faster than Block Rams. It serves as a relatively large memory structure (i.e. larger than distributed RAMs or a bunch of D-Flip-flops grouped together, but much smaller than off chip memory resources).

4.2.8 DSP Cores

Digital Signal Processors (DSPs), as shown in Figure 4-9 , are another common type of core that is offered as an IP core or an embedded core. These are essentially specialized processors that are used for manipulating analog signals. They are commonly used for filtering and compression of video or audio signals, Multiply Accumulate block or MAC is implemented as DSP slice and MAC is mainly used as a building block for complex DSP applications.

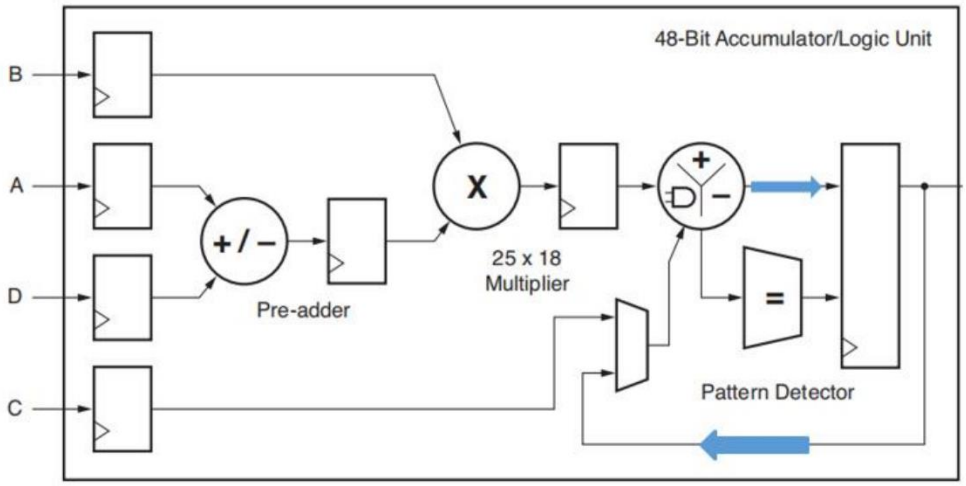


Figure 4- 9 DSP block diagram

4.2.9 Embedded Cores

The embedded core will be optimized for the vendor's process to give good timing and power consumption numbers. The core will be placed as a single cell on the silicon die and so the performance of the core will not depend on the rest of the design since it won't need to be placed and routed.

4.2.10 Special I/O Drivers

Special I/O drivers are also being embedded into programmable devices. The newer buses inside personal computers need to have very tightly controlled timing and must be driven by special high-drive, impedance-matched circuits. The I/O buffers need to have inputs with very specific voltage threshold values.

4.2.11 FPGA Design Flow

Figure 4-10 shows the steps of the design flow.

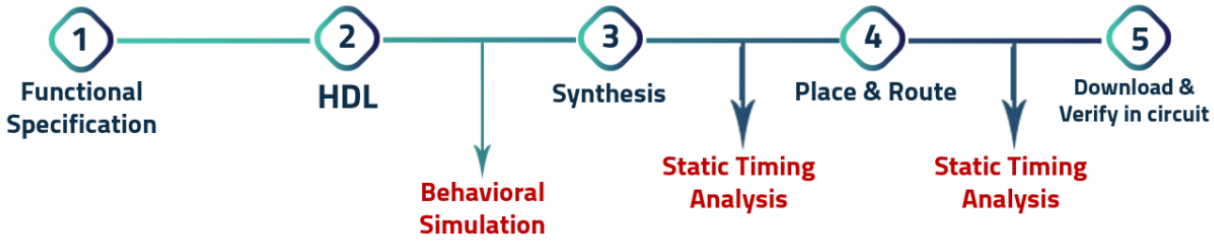


Figure 4- 10 FPGA Design flow

1-Functional Specifications: in this step, all specifications for the application are determined along with good understanding of function of this application.

2-HDL: the HDL code that describes that function is written, and then Behavioral Simulation is done to make sure that the HDL describes the function needed correctly.

3-Synthesis: HDL is converted into logic gates and other cells present in the FPGA itself, Static timing analysis is done to approximately calculate the maximum clock delay of the application and calculate the maximum clock speed achieved for the application.

4-Place & Route: The logic blocks and cells in the FPGA are connected together, and Static Timing Analysis is done again to calculate the exact delay model of the application.

5-Download & Verify in circuit: The HDL code is burned on the FPGA

4.2.12 Fixed Point Background

Deep convolutional neural network (CNN) inference requires significant amount of memory and computation, which limits its deployment on embedded devices. To alleviate these problems to some extent, prior research utilize low precision fixed-point numbers to represent the CNN weights and activations. However, the minimum required data precision of fixed-point weights varies across different networks and also across different layers of the same network. A fixed-point representation of a number consists of integer and fractional components and sign bit as shown in Figure 4-11, where WL represents word length, S represents the sign bit, I represent the integer bits and F represents the fractional bits. With this representation the range of numbers is $[2^{-I}, 2^{I}]$, and a step size (resolution) of (2^{F}).

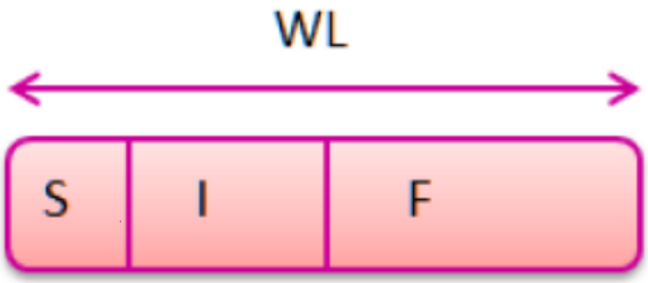


Figure 4- 11 Word line

4.2.13 Number representation

There are two representations for numbers in bits form and we used the 2's complement for our design

1- Signed magnitude:
   - ❖ Has 2 representations for zero (± 0)

2- 2's Complement:
   - ❖ Addition and subtraction still 2's complement
   - ❖ Multiplication needs sign extension to have the same representation after multiplication

4.2.14 Fixed point multiplication

Fixed-point multiplication is the same as 2's compliment multiplication but requires the position of the "point" to be determined after the multiplication to interpret the correct result. The determination of the "point's" position is a design task. The actual implementation does not know (or care) where the "point" is located. This is true because the fixed-point multiplication is exactly the same as a 2's complemented multiplication, no special hardware is required.

4.2.15 Kernel Storage

4.2.15.1 FPGAs Read Only Memory

Some FPGAs have off-chip ROMs, which have great utility in account of high latency; however, our model implementation targets speed as its first priority, so, off chip memories would not be an option. FPGAs hold two types of memories, any of which could fit for the model parameters.

4.2.15.2 BRAMs

FPGAs normally have on-chip BRAM matrix, which could be configured as FIFO, RAM or ROM. Targeted device (Virtex-7 x690t) contains a sum of 2940 BRAM 18Kb instances, each can be configured to 4Kb x 4, 8Kb x 2 or 16Kb x 1. BRAMs can have dual ports for the same instances, allowing performance of half the latency.

4.2.15.3 Distributed ROM

Xilinx FPGAs offer another type of memory, which is LUT, distributed ROM, which can be configured to hold design parameters. Each LUT can be configured as 6 input 1-bit ROM (64x1). Normally, distributed ROMs grant more speed than BRAMs, which makes them a good approach to the model implementation.

4.2.15.4 Initializing ROM

ROMs are usually initialized with Verilog system task $readmemb/ $readmemh inside initial procedural block, which loads memory contents from a file, specifying start and end addresses.

4.2.15.5 ROM Design

For the weights memory implementation, the approached implementation is to design separate 2D ROM arrays (discarding the bits dimension) accessed as a 3D memory for each filter in the design

4.3 Hardware Methodology

Because our model uses a pre-processing function that is STFT, it has become a necessity to design a digital model to implement the STFT operation

4.3.1 STFT Digital implementation

In the STFT hardware implementation, we need to first pad the frame of the input signal to be 64 points instead of 32 points to fit the size of the input of STFT that we used in our model. According to our assumptions that the input signal will be stored in internal memory at FPGA as well as the values of the Hanning window, and we will loop on them with a counter with 6 bits. So, we put a condition if the counter of the Hanning window exceeds 32 points stop the input signal and put zero instead of it until the counter resets itself by counting to zero. Then starts the new loop with the rest of the signal with the same condition. But when we try to benefit from these cycles with zeros, we found that the speed of the multiplier will increase more than the speed of the FFT. Because the FFT gets each input point for each clock cycle.

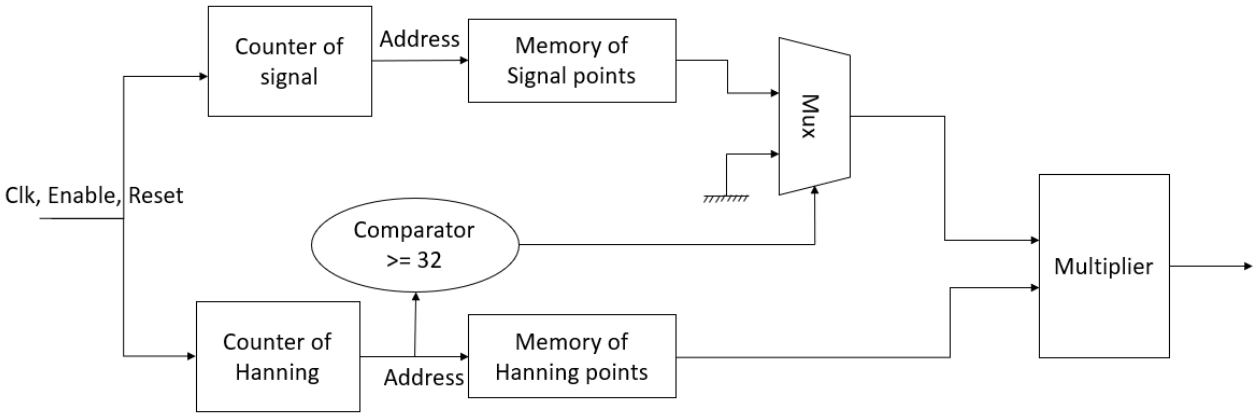
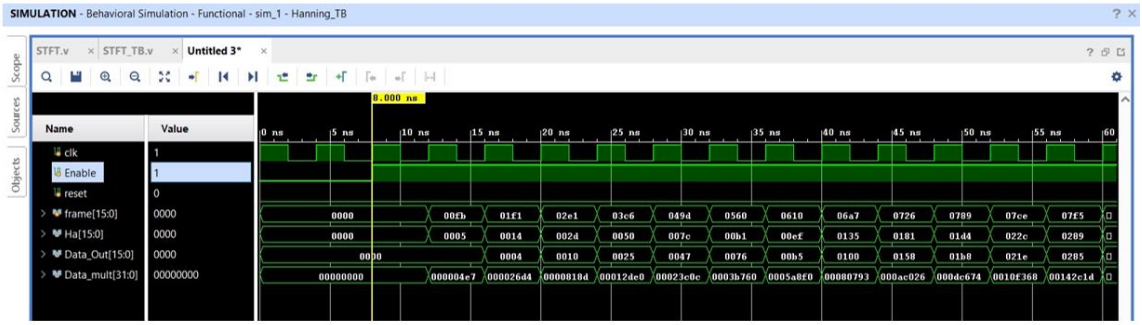


Figure 4- 12 Schematic of Hanning Multiplier

After Padding the input frame, we multiply the frame with the Hanning window (as we know the output of the multiplier bits increases double of input bits 2N, so we crop the output to be



Sign bit = 1, integer bits = 4, fraction bits = 11

Signals	1st	2nd	3rd	4th	5th	6th	7th	8th
Frame	0.00	0.123	0.243	0.36	0.472	0.577	0.672	0.758
Hanning	0.00	0.002	0.01	0.022	0.039	0.061	0.087	0.117
Output of multiplier	0.00	0.0003	0.0024	0.0079	0.0184	0.0349	0.0581	0.0884
Output of Hanning block	0.00	0.000	0.002	0.008	0.018	0.035	0.058	0.088

Figure 4- 13 Example of output of multiplier with numbers

N bits to decrease the complexity) and then enter it to the FFT 64 points. The FFT that we used is IP core provides four different architectures that offer a trade-off between core size and transform time. So, we choose the best throughput because we care about performance rather than resources.

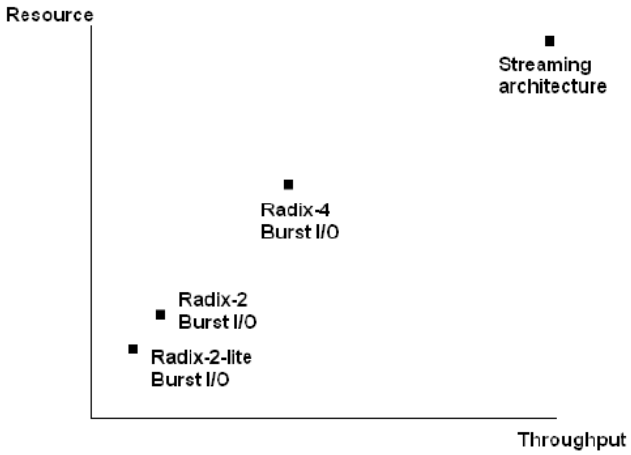


Figure 4- 14 Resources VS Throughput for architecture options

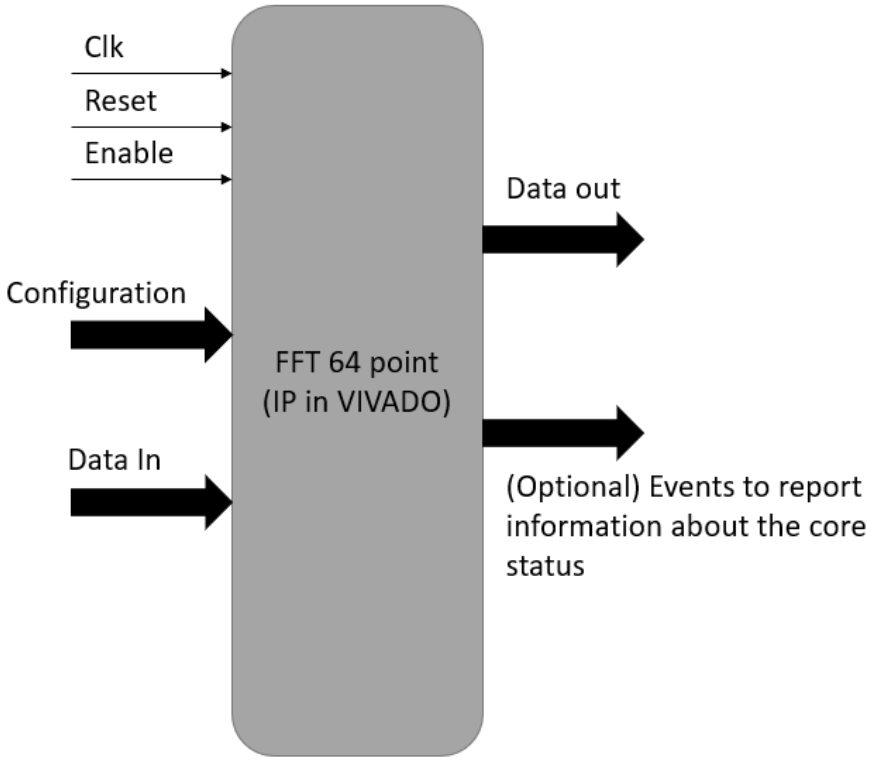
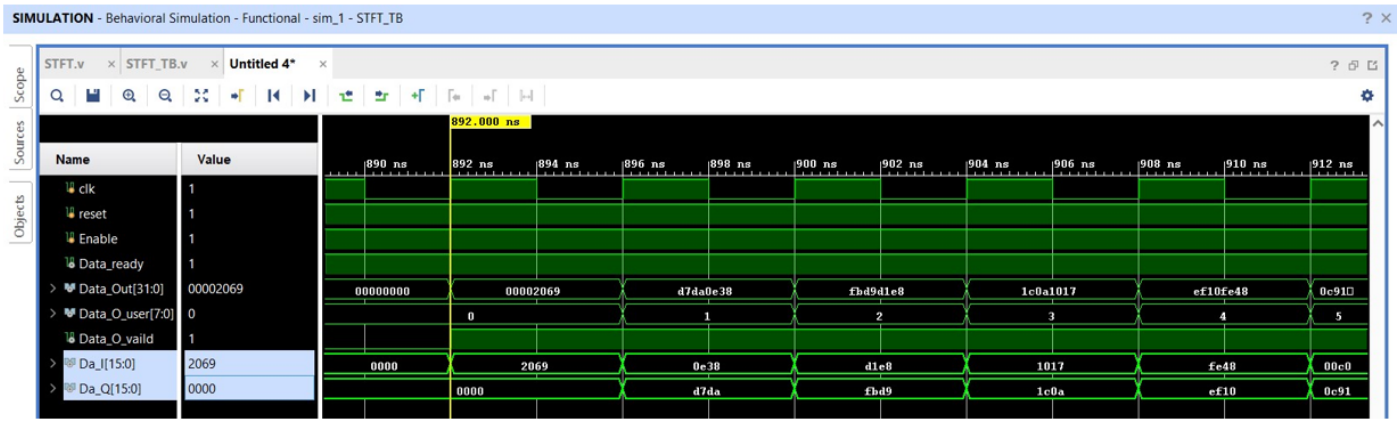


Figure 4- 15 FFT IP block in vivado

The FFT IP core is generic, it can be IFFT or FFT. This is determined by configuration signals that are input to the block. The input data frame enters point by point but at the end of each frame of 64 points, we should send a pulse to inform the core that this point is the last point of the frame. So, to handle this pulse, we put a flag that rises when the counter is equal to 63 and drops when the counter is equal to 0. And there is an output signal that informs the user that the FFT is ready to accept new input. The main signals are rising-edge clock (aclk), reset (areset) and active-high clock enable (aclken). And the output port exits the result of the FFT point by point with its index and signal valid to receive the output. There are two ports optional to use, so we take one of them which is called event with 6 signals to report information about the core status.



Sign bit = 1, integer bits = 4, fraction bits = 11

Signals	1st	2nd	3rd	4th	5th	6th	7th	8th
Output of Hanning block (Input FFT)	0.00	0.00	0.002	0.008	0.018	0.035	0.058	0.088
Output real	4.0513	1.7773	-5.7617	2.0112	-0.2148	0.0938	0.1050	-0.1528
Output imaginary	0.00	-5.0186	-0.5190	3.5049	-2.1172	1.5708	-1.2534	1.0425

out_fft_mat

64x1 complex double

	1
1	4.0560 + 0.0000i
2	1.7750 - 5.0208i
3	-5.7602 - 0.5165i
4	2.0108 + 3.5024i
5	-0.2149 - 2.1149i
6	0.0952 + 1.5683i
7	0.1014 - 1.2526i
8	-0.1489 + 1.0432i
9	0.2116 - 0.8900i
10	-0.2344 + 0.7722i

Figure 4- 16 Example of output of FFT with numbers

4.3.2 Model overview

Figure 4-12 below shows the model we designed to be implement on FPGA and it shows the number of parameters, number of operations, number of DSP's needed for each layer and how much clocks each layer will take to compute its output feature map.

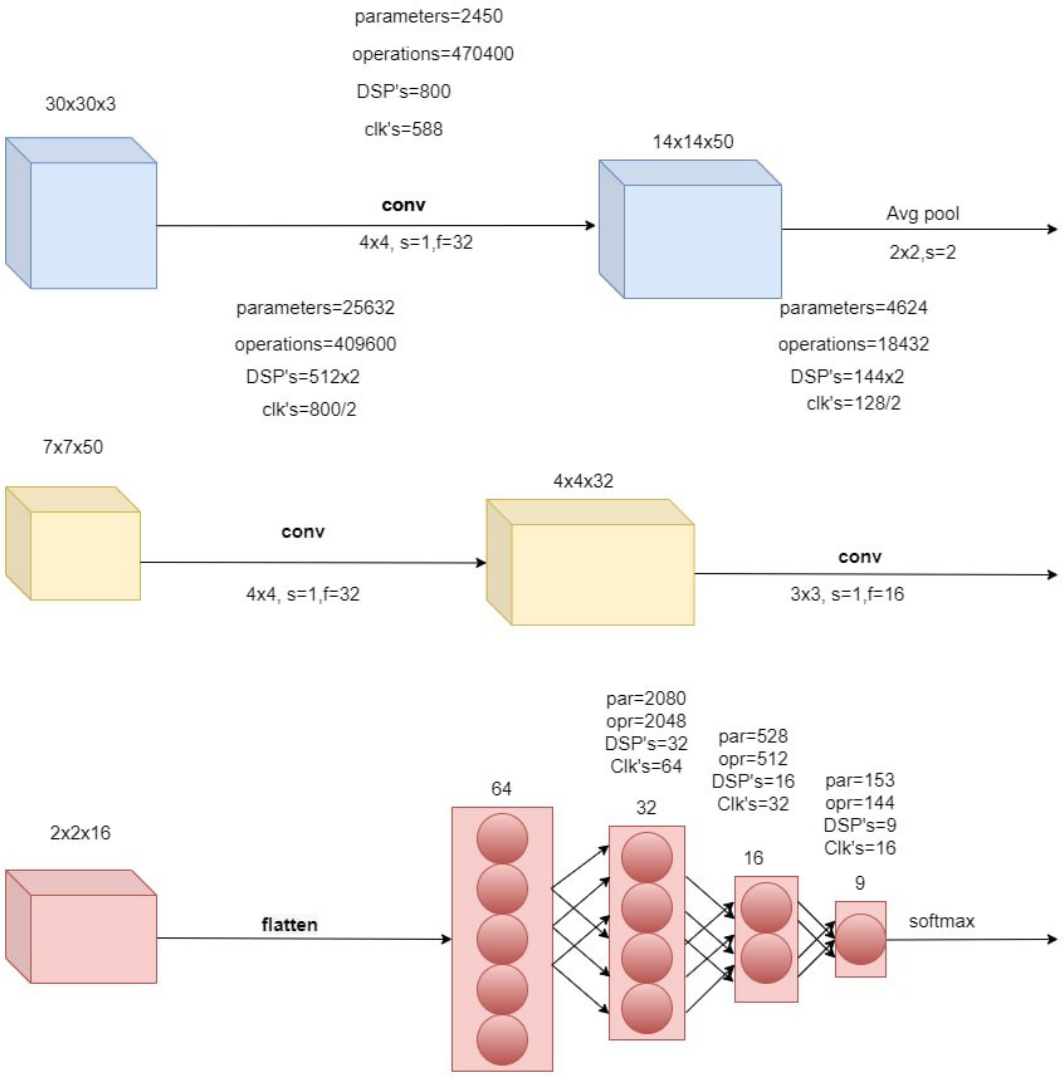


Figure 4- 17 Block diagram for the implemented Model

Our design aim is speed and accuracy in a trade off with area and power since the need of continuous monitoring of the spectrum requires a very fast model implementation with high throughput and low latency so according to these specifications we modeled all the CNN layers with multiplication addition tree (MAT) implantation since it is way more faster than using MAC's and for fully connected (FC) layers we have chosen the multiplication and accumulation (MAC) implementation since implementing FC layers with MAT in our model is not efficient and will not increase the throughput of the implementation since the bottleneck is already at the first CNN layer.

4.3.3 Summarization table for the model

Noting that the number of DSP's and Clocks here is calculated based on the design and we will discuss each layer separately later in this literature.

Table 4- 1 Digital Model parameters

Layer name	Parameters	Multiply Operation	DSP's	Clocks
Conv 1	2,450	470,400	800	588
Average pooling	0	0	0	49
Conv 2	25,632	409,600	1024	400
Conv 3	4,624	18,432	288	64
FC 1	2,080	2048	32	64
FC 2	528	512	16	32
FC 3	153	144	9	16
Total	35,467	901,136	2,169	1213

4.3.4 MAC: Multiplication and Accumulation

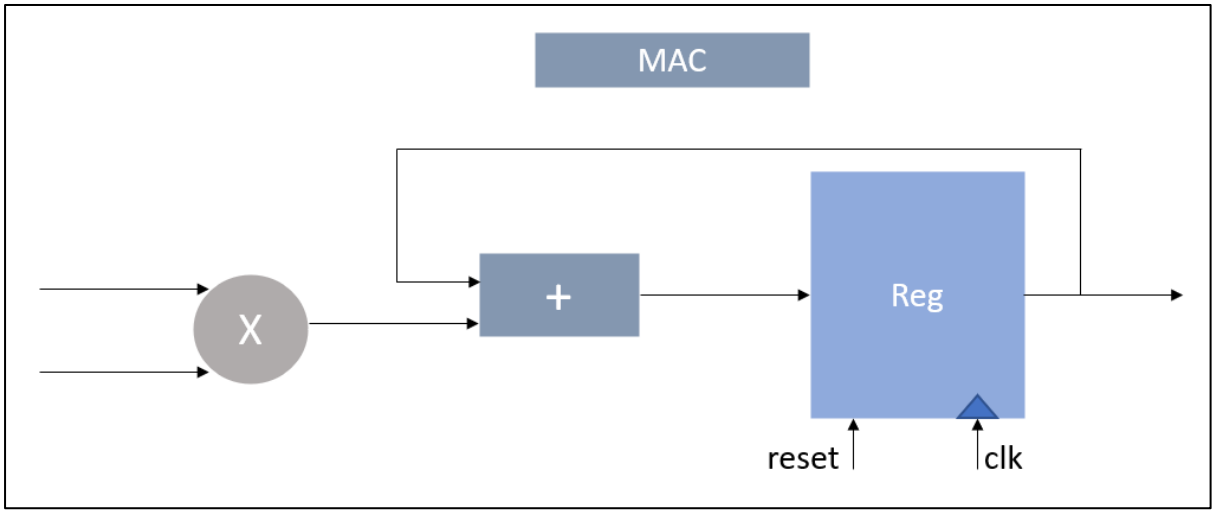


Figure 4- 18 MAC block diagram

Figure 4-18 shows internal design of one MAC (which corresponds to one DSP) that consists of a multiplier, accumulation register, adder and reset signal used to reset the register with the bias value after each output. The number of MACs or DSPs per FC layer depends on the number of neurons in the layer.

4.3.5 MAT: Multiplication Addition Tree

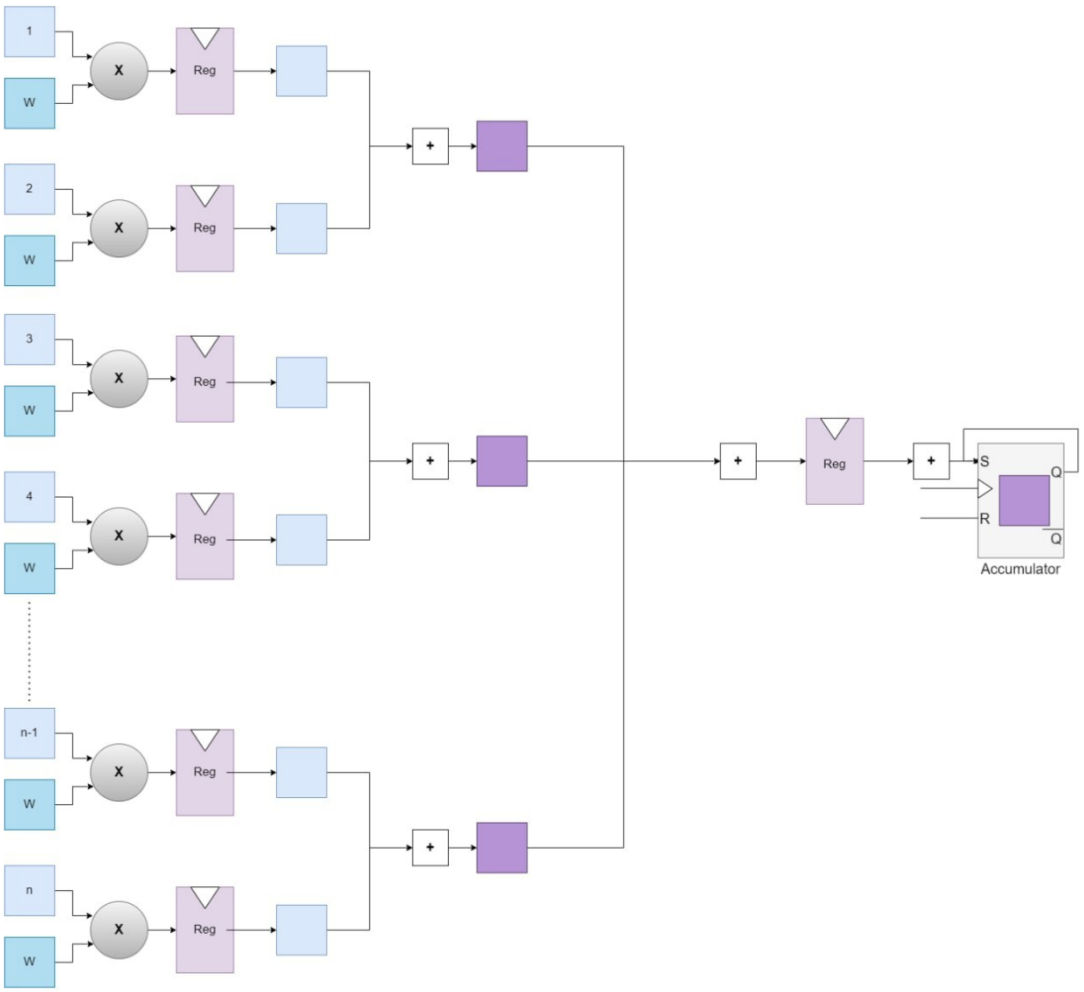


Figure 4- 19 MAT block diagram

Figure 4-19 shows internal design of one MAT which correspond to number of DSP's equal to the number of inputs as an example MAT16 will have 16 input data and 16 input weights so the number of DSP's for it will be 16 DSP and the number of registers in the data path will equal to ceiling($log_2 16$) = 4 registers, and an accumulator will be placed after it since the MAT will have only one or two channels as an input at each clock so it will need to accumulate the values until all the channel enter the MAT and it also have reset signal to reset the accumulator with the bias value.

4.3.6 Average Pooling layer implementation

Usually in CNN algorithm, convolutional layers are interleaved with pooling layers to reduce the dimensions of its input feature maps retaining the most dominant information. These layers may be average or maximum pooling layers, for our model there were only one average pooling layer.

Figure 4-20 shows the implementation of the average pooling in out model, we benefited from the size of the pooling layer and replaced the division by 4 with only 2-bits shift at the input

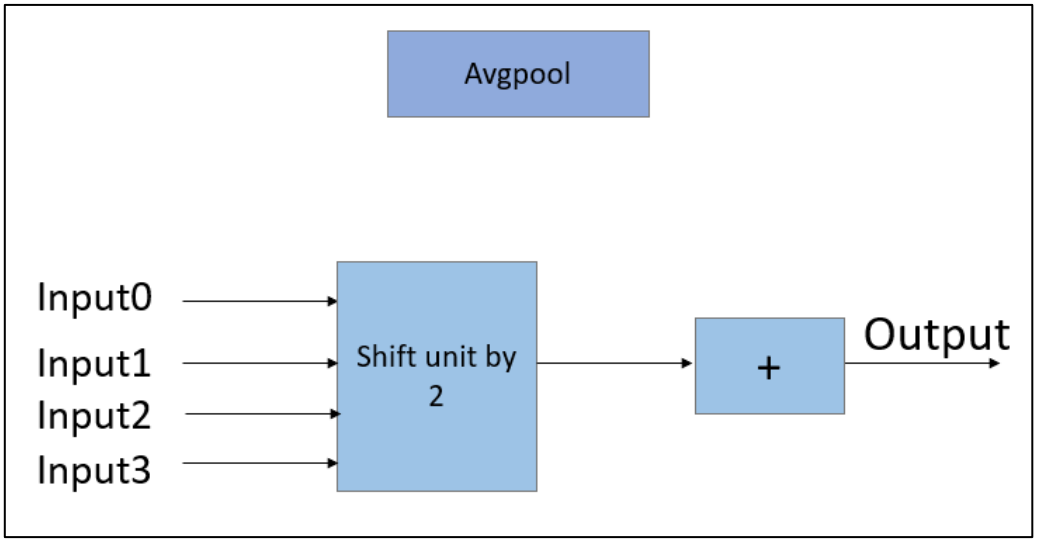


Figure 4- 20 Average pool block diagram

4.3.7 Output prediction implementation (soft max)

For the soft max block we have 9 classes and need to output the address of the class that correspond to the maximum value, we implemented it using 2 stages of comparing, first one compare each 3 inputs together and gives the maximum on of each 3 inputs and there addresses then the second layer compare the 3 maximums again and output the address of the maximum number.

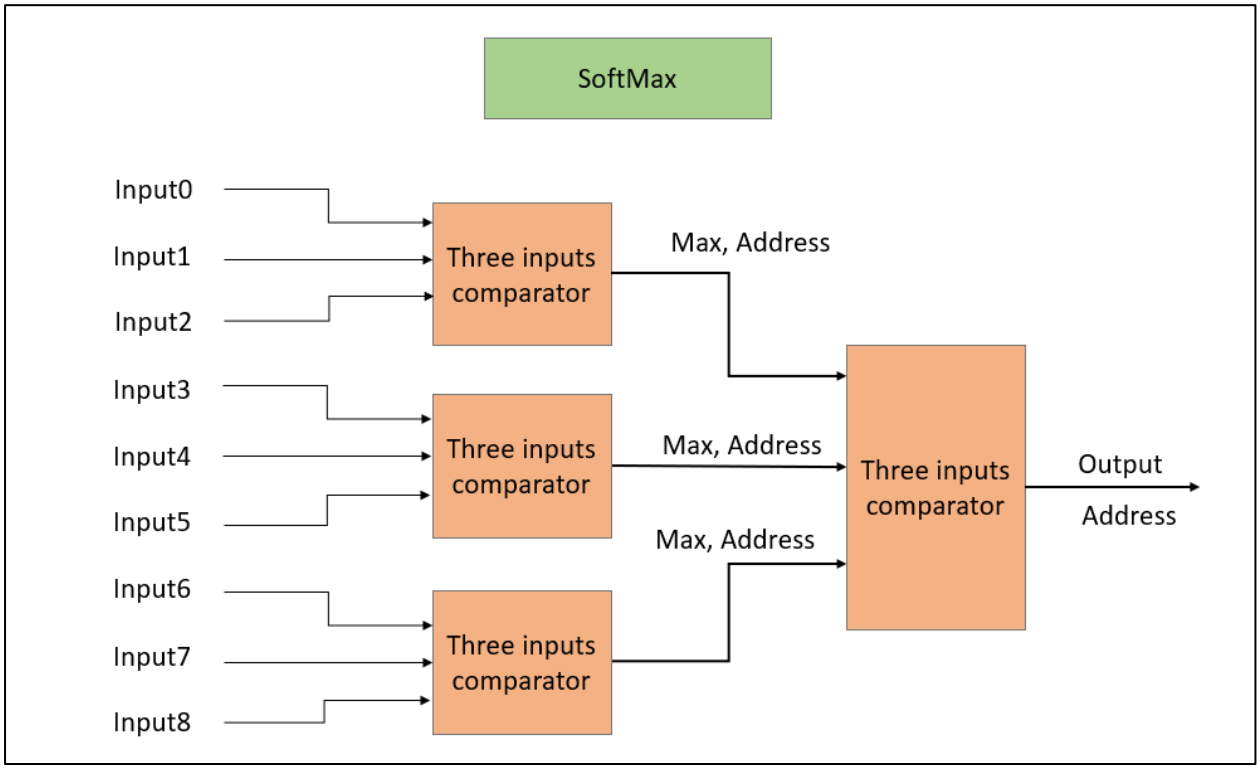


Figure 4- 21 Softmax implementation block diagram

4.3.8 Layer 1 & 2 (conv1 and average pooling)

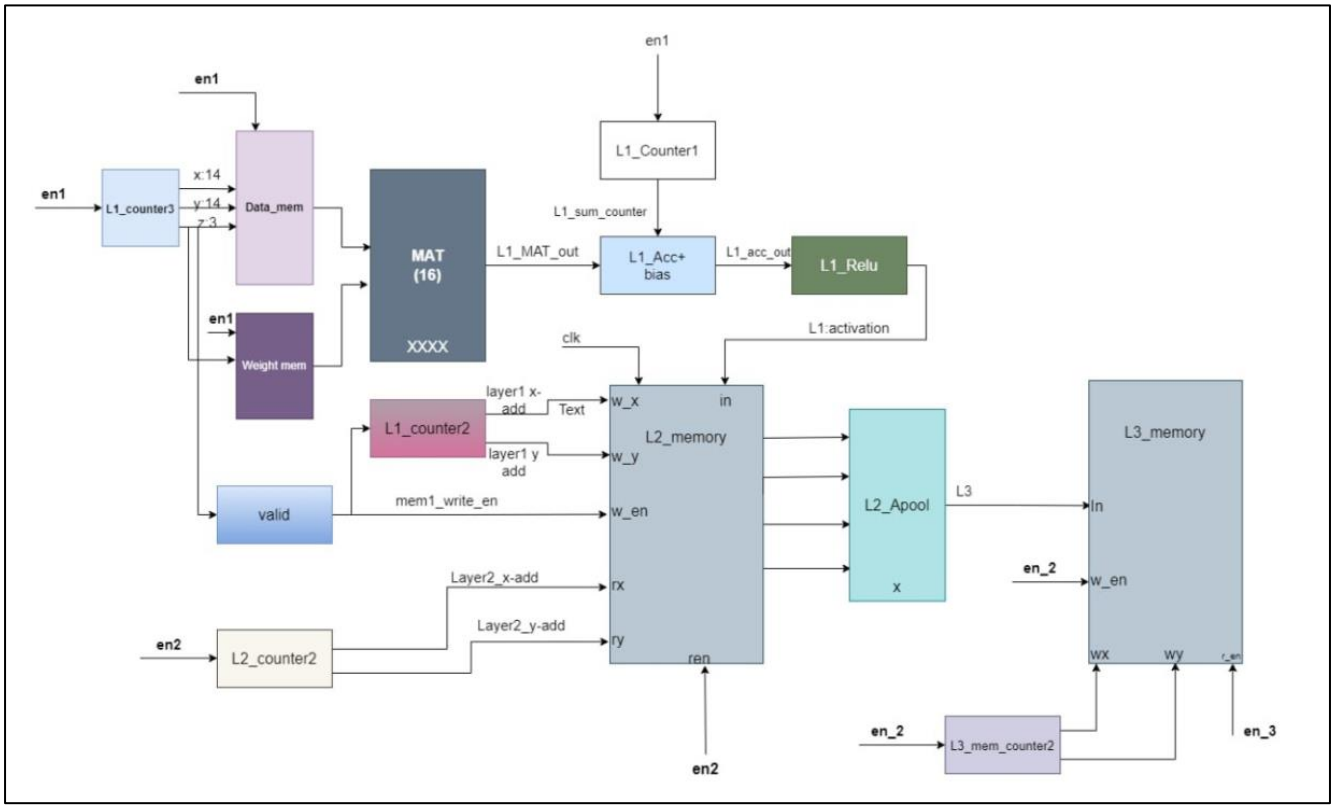


Figure 4- 22 Convolution layer digital implementation block diagram and avgpool

Figure 4-22 shows the implementation of layer 1 and 2 which consists of data memory to hold the values of the STFT output and weight memory to hold the weights values for layer 1 then we used a MAT block with 16 input from the data memory and 16 input from the weight memory which corresponds to a one channel of the filter then to calculate the 4×4×3 filter output we parse through the channels and accumulate the value and apply the Relu activation function by a simple multiplexer with selection line based on the sign bit and when we reach the last channel the value accumulated will be written in the next layer memory and the accumulator will get reset with the bias value so it will take three clocks to calculate each point in the output feature map since there are three channels in the data memory taking in care that accumulating will start after the MAT pipeline is filled so the number of clocks needed for this layer is (3 channels×14×14) = 588 clocks

Then after L2_memory is filled layer 2 start to read 2×2 values each clock and calculate the average value and gives it to the next layer memory.

This is generated for each filter in layer 1, which is 50 filters so the L3_memory will hold 7×7×50 values.

4.3.9 Layer 3 (conv2)

Figure 4-23 shows The same approach used in layer 1 is used in layer 3 but with changes depending on the sizes of layer 3 and to be as close as possible to real time implementation. The input shape of this layer is 7×7×50 and the output feature map is 4×4×32 so if the same implementation is done here this layer will take 4×4×50 = 800 clocks

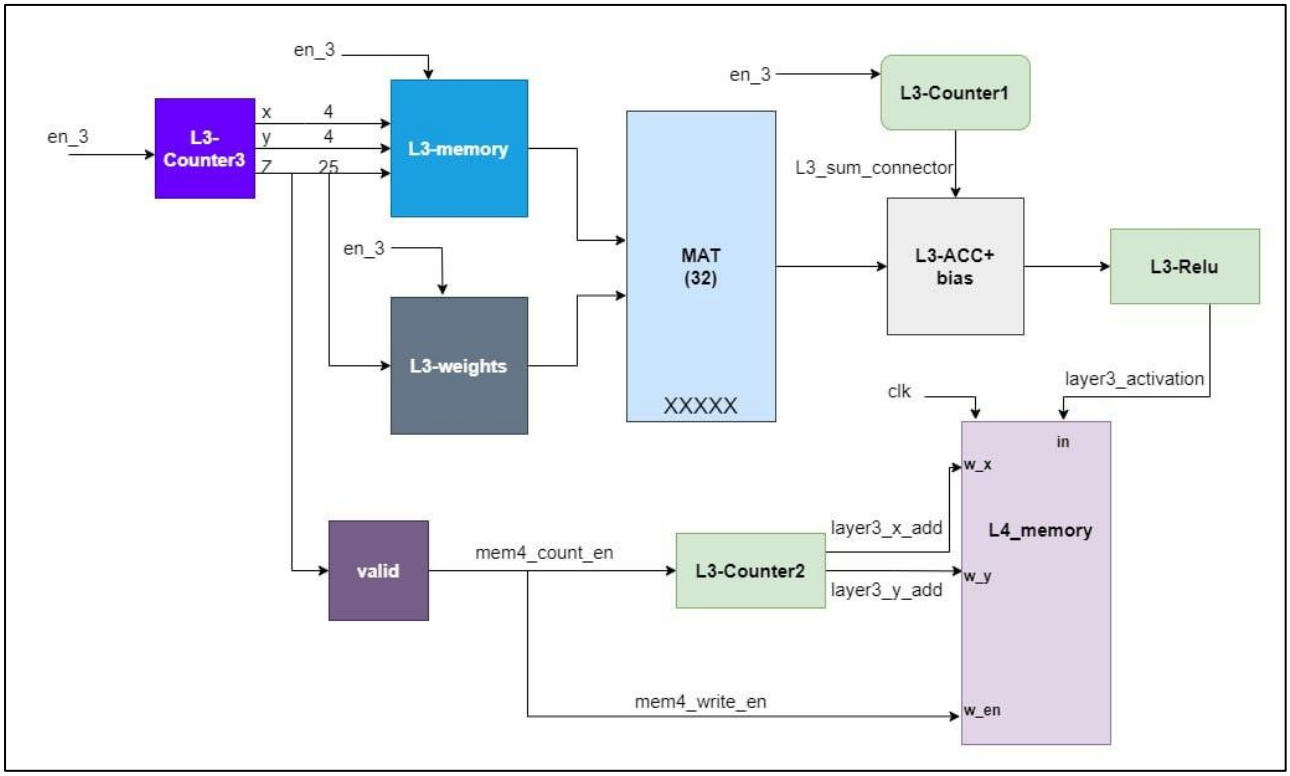


Figure 4- 23 Block diagram of the digital implementation of the second convolution layer

It will take 16×32 = 512 DSP's which will create a bottleneck to the architecture in this layer so the approach here is instead of calculating 1 channel every clock we calculate 2 channel every clock which will reduce the number of clocks need to the half and multiply number of DSP's need by 2 so with this approach the number of clocks needed will be 4×4×50/2 = 400 clock with 32×32 = 1024 DSP slices for only this layer.

4.3.10 Layer 4 (conv3)

Figure 4-24 shows the same approach used in layer 3 is used in layer 4 but with changes depending on the sizes of layer 4

In this layer the input feature map is 4×4×32 and the output is 2×2×16

Since the number of DSP slices in the intended Virtex-7 FPGA is large and we still have more DSP's than the total needed until now plus the relative small size of this layer, for all those reasons we decided to use the same approach used in layer 3 of calculating 2 channels every clock so the number of clocks needed will be 2×2×32/2 = 64 clock with 18×16 = 288 DSP slices

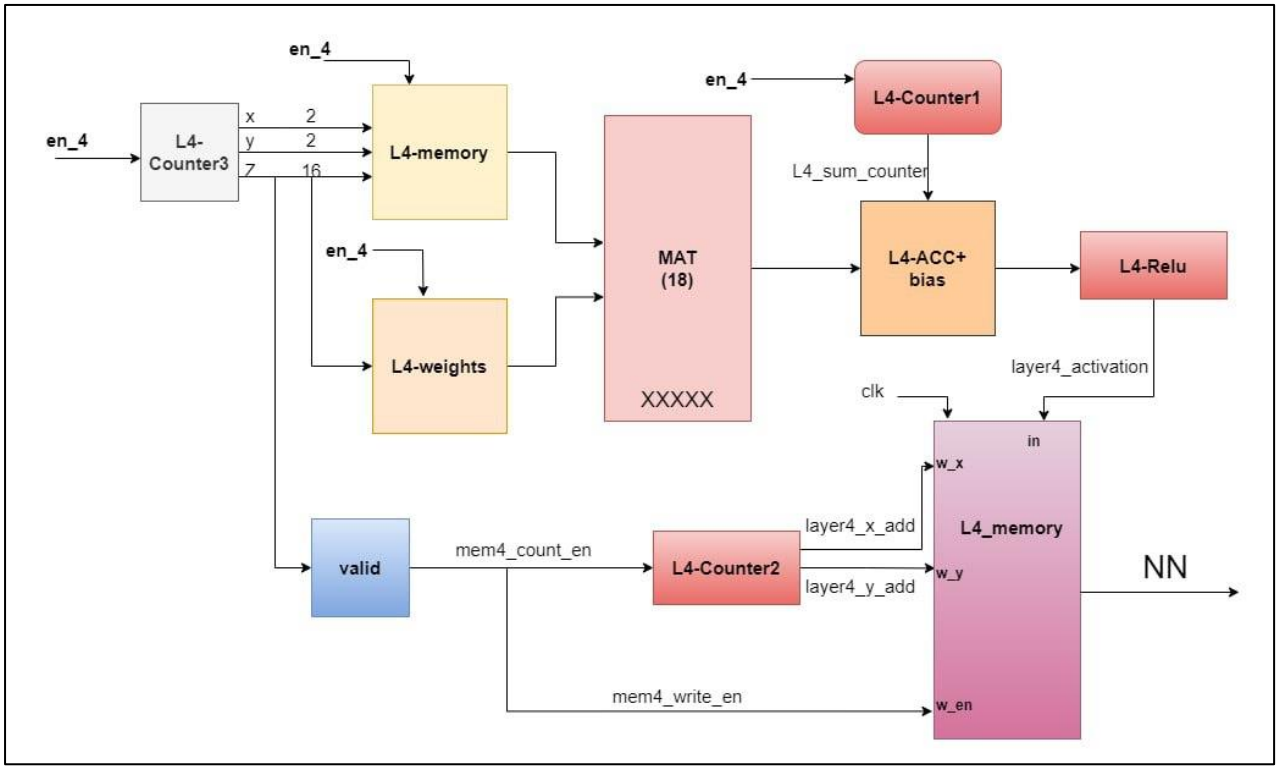


Figure 4- 24 Block diagram of the digital implementation of the third convolution layer

4.3.11 Fully Connected Implementation

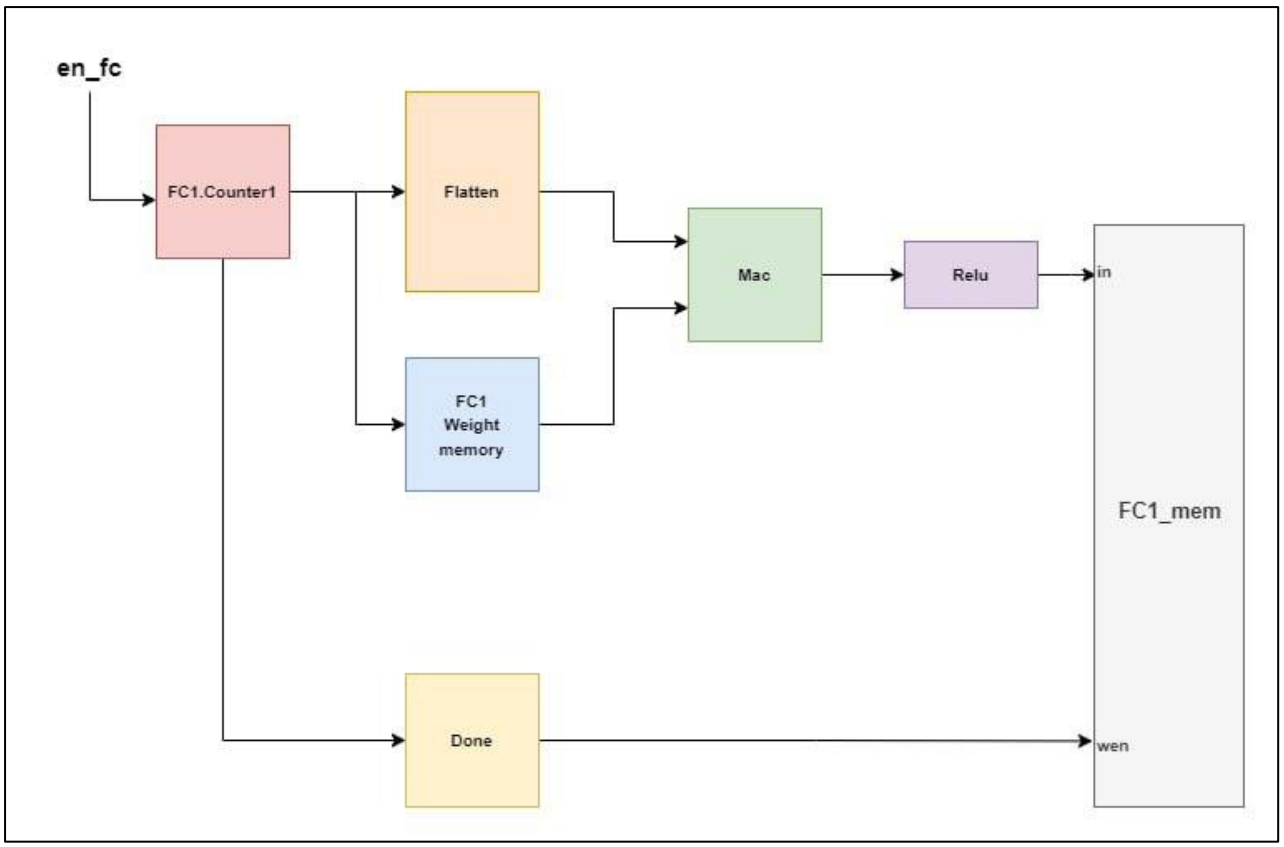


Figure 4- 25 Fully Connected layer digital implementation block diagram

Figure 4-25 shows the implementation of the fully connected layer which is consider to be more straight forward compared to the Convolutional layers.

In this architecture, we used MAC blocks to calculate the neurons output all at the same time so the output memory will be accessed and filled only one time in the evaluation process.

At each clock cycle after the enable is activated the MAC will get the value of the data and weight and accumulate it until all the data are given to it then the MAC is reseted to the value of the bias and hold until the next enable is activated.

This implementation is applied to all the FC layers with a controller take a fire signal and output all the enables and reset signals to all the FC layers.

Then at the last FC layer have soft max activation function that gives the final output class of the model.

4.3.12 Behavioural simulation example

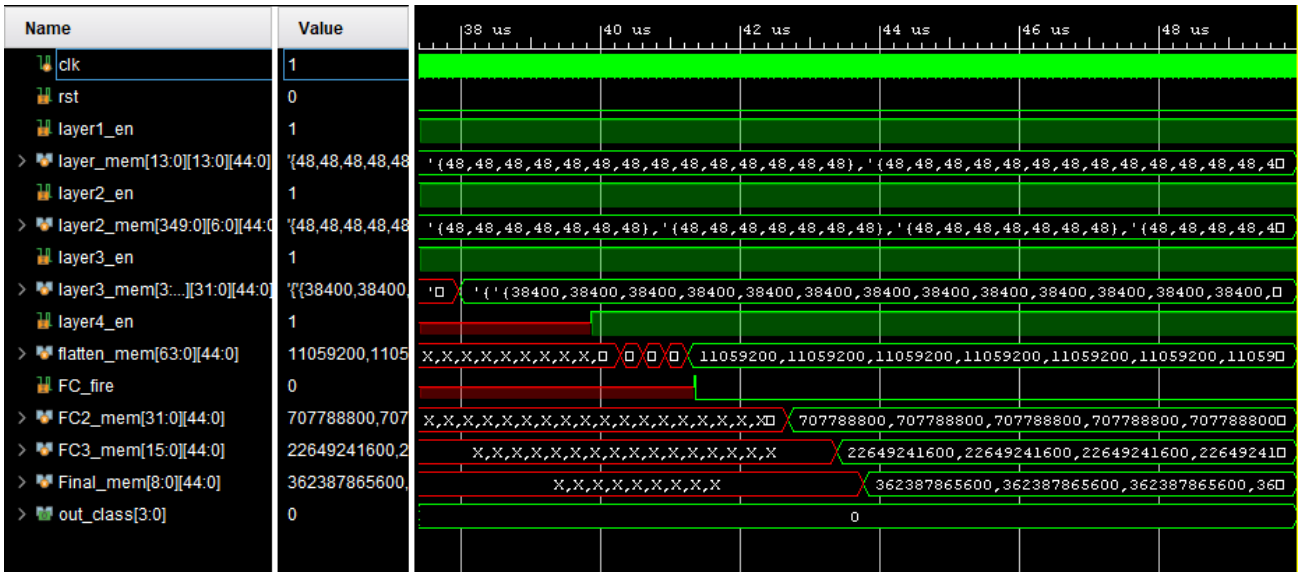


Figure 4- 26 an example behavioral simulation results

Figure 4-26 shows the memories of the model being filled in a behavioural simulation of our design with all weights is initialized to 1 and all biases initialized to 0, with zero bits for fraction part and no limitation on number of bits needed just for testing purposes

So for verification of this part we can calculate the values manually and compare it with the results shown in figure 4-26

Conv 1 output = 4×4×3 = 48

Average Pool output = Conv 1 output (since all the values are equal)

Conv 2 output = Conv 1 output ×4×4×50 = 38400

Conv 3 output = Conv 2 output ×3×3×32 = 11059200

FC1 output = Conv 3 output ×64 = 7077888000

FC2 output = FC1 output ×32 = 22649241600

FC3 output = FC2 output ×16 = 362387865600

Which are the same values showed in the behavioural simulation, also here we can see that the default value for the out class is class 0 which corresponds to the normal class.

Future work

Communication Part:

- Collect real data instead of data generated using simulation

Deep Learning Part:

- Use wavelet transform instead of STFT
- Test the model on collected data instead of simulation
- Train the model on OFDM dataset we generated

Digital Part:

- Implement the model on FPGA
- Verify the implementation we designed on FP

References

F. N. Iandola, M. W. Moskewicz and K. Ashraf, "SqueezeNet: AlexNet-level accuracy with 50xfewer parameters and

S. Han, et al. "DSD: Dense-sparse-dense training for deep neural networks.", 2016.

S. Sombatsiri, et al. "Parallelism-flexible Convolution Core for Sparse Convolutional Neural Networks.", 2018.

A. Khan, A. Zahoora, and A. Qureshi," A Survey of the Recent Architectures of Deep Convolutional Neural Networks", 2019

M. Kulin, T. Kazaz, I. Moerman and E. De Poorter, "End-to-End Learning From Spectrum Data: A Deep Learning Approach for Wireless Signal Identification in Spectrum Monitoring Applications," in IEEE Access, vol. 6, pp. 18484-18501, 2018.

A. Emad, H. Mohamed, A. Farid, M. Hassan, R. Sayed, H. Aboushady, H. Mostafa, Deep Learning Modulation Recognition for RF Spectrum Monitoring

Nistha Tandiya, Ahmad Jauhar, Vuk Marojevic, Jeffrey H. Reed**,** Deep Predictive Coding Neural Network for RF Anomaly Detection in Wireless Networks

RADIOML 2016.10A Dataset [online]. Available: https://www.deepsig.ai/datasets.

Xilinx Vivado Design Suite. Available at: https://www.xilinx.com/products/design-tools/vivado.htm.

Xilinx Vivado Design Suite. Available at: https://docs.xilinx.com/viewer/book-attachment/jKn_d6EeSeSm4b25FBbCOA/KtJ2q9rEZvP4hqj3BbbU_Q

https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05

https://www.techtarget.com/searchmobilecomputing/definition/spectrum-efficiency

https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

https://www.sciencedirect.com/topics/computer-science/digital-modulation

https://en.wikipedia.org/wiki/Pulse_shaping

https://www.projectrhea.org/rhea/index.php/Upsampling_Slecture_Molveraz

https://en.wikipedia.org/wiki/Rician_fading