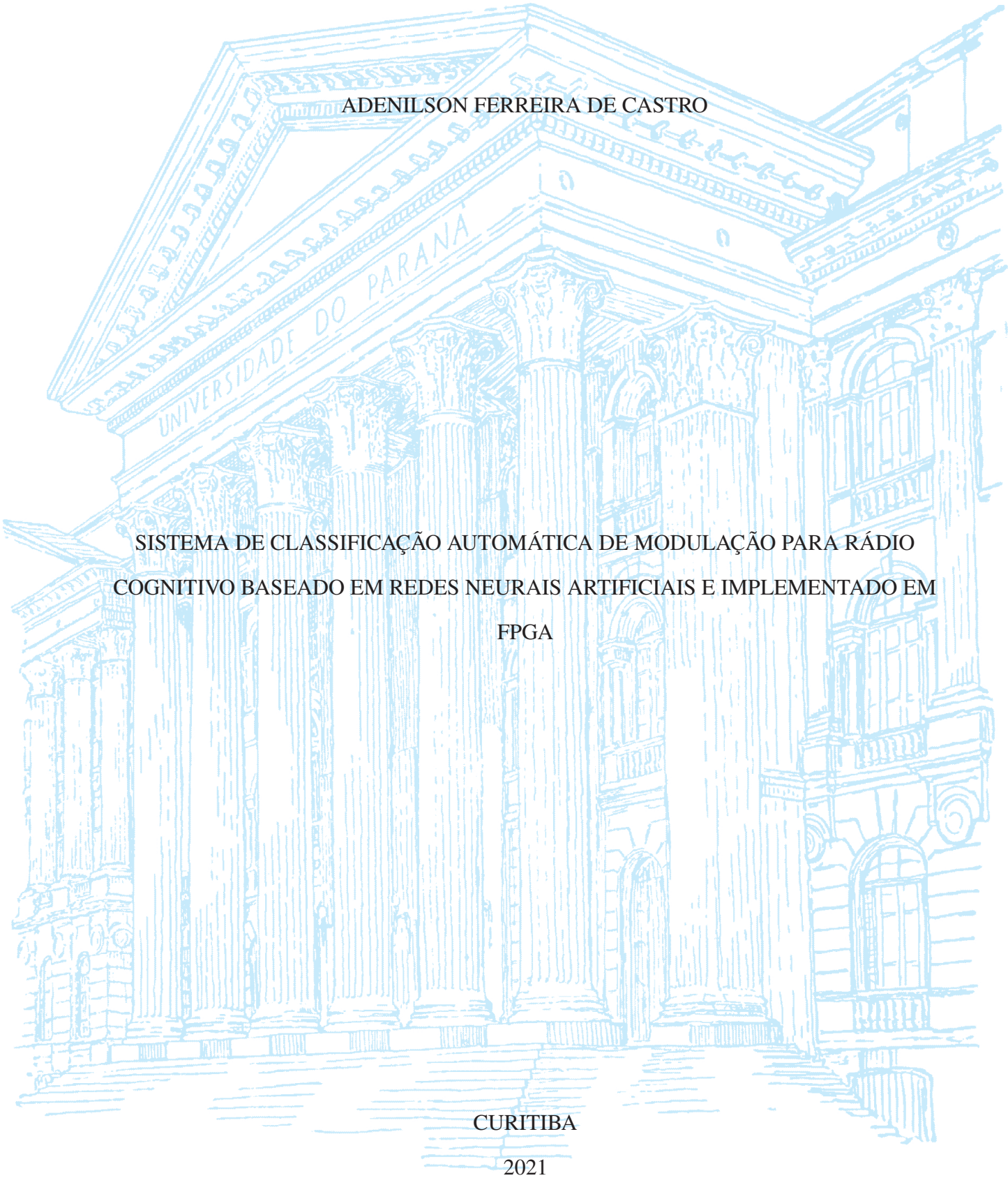
UNIVERSIDADE FEDERAL DO PARANÁ

ADENILSON FERREIRA DE CASTRO



SISTEMA DE CLASSIFICAÇÃO AUTOMÁTICA DE MODULAÇÃO PARA RÁDIO COGNITIVO BASEADO EM REDES NEURAIS ARTIFICIAIS E IMPLEMENTADO EM FPGA

CURITIBA

2021

ADENILSON FERREIRA DE CASTRO

SISTEMA DE CLASSIFICAÇÃO AUTOMÁTICA DE MODULAÇÃO PARA RÁDIO COGNITIVO BASEADO EM REDES NEURAIS ARTIFICIAIS E IMPLEMENTADO EM FPGA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica no Programa de Pós-Graduação em Engenharia Elétrica, Setor de Tecnologia, da Universidade Federal do Paraná.

Orientador: Prof. Dr. Luis Henrique Assumpção Lolis.

Coorientador: Prof. Dr. André Augusto Mariano.

CURITIBA

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

C355s Castro, Adenilson Ferreira de
Sistema de classificação automática de modulação para rádio cognitivo baseado em redes neurais artificiais e implementado em FPGA [recurso eletrônico] / Adenilson Ferreira de Castro. – Curitiba, 2021.

Dissertação - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, 2021.

Orientador: Luis Henrique Assumpção Lolis.
Coorientador: André Augusto Mariano.

1. Modulação (Eletrônica). 2. Redes neurais (ciência da computação). 3. Redes de rádio cognitivas. I. Universidade Federal do Paraná. II. Lolis, Luis Henrique Assumpção. III. Mariano, André Augusto. IV. Título.

CDD: 006.32

Bibliotecária: Vanusa Maciel CRB- 9/1928

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA ELÉTRICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **ADENILSON FERREIRA DE CASTRO** intitulada: **SISTEMA DE CLASSIFICAÇÃO AUTOMÁTICA DE MODULAÇÃO PARA RÁDIO COGNITIVO BASEADO EM REDES NEURAIS ARTIFICIAIS E IMPLEMENTADO EM FPGA**, sob orientação do Prof. Dr. LUIS HENRIQUE ASSUMPÇÃO LOLIS, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 06 de Abril de 2021.

Assinatura Eletrônica
06/04/2021 15:51:07.0
LUIS HENRIQUE ASSUMPÇÃO LOLIS
Presidente da Banca Examinadora

Assinatura Eletrônica
07/04/2021 08:15:43.0
EVELIO MARTÍN GARCÍA FERNÁNDEZ
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
06/04/2021 15:54:51.0
GLAUBER GOMES DE OLIVEIRA BRANTE
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

Assinatura Eletrônica
06/04/2021 16:51:22.0
EDUARDO GONÇALVES DE LIMA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Perdi alguma coisa que me era essencial, e que já não me é mais. Não me é necessária, assim como se eu tivesse perdido uma terceira perna que até então me impossibilitava de andar mas que fazia de mim um tripé estável. Essa terceira perna eu perdi. E voltei a ser uma pessoa que nunca fui. Voltei a ter o que nunca tive: apenas as duas pernas. Sei que somente com duas pernas é que posso caminhar. Mas a ausência inútil da terceira me faz falta e me assusta, era ela que fazia de mim uma coisa encontrável por mim mesma, e sem sequer precisar me procurar.

Clarice Lispector.

ACKNOWLEDGEMENTS

I thank God for the miracle of life.

I thank my parents for being the support I needed when times were tough, unconditional love, and always believing in my dreams.

I thank my grandmother, Justina da Silva, for her love and for always stimulating my curiosity. I hope that wherever you are, I can make you proud. I also thank my grandfather, Valdir Castro, and my aunt Célia Castro, who died during the journey of this research. The memories we constructed together will always be one of the foundations where I find my strength to keep moving forward.

I thank my advisor, Luis Lolis, for the time, patience, guidance, and for helping me to make a personal dream come true. A special thanks to my co-advisor, André Mariano, for allowing me to overcome this challenge and always keeping me motivated to become a better Engineer and human being.

Thanks to all the research group colleagues, the GICS, for the long conversations, advice, laughs, and the many cups of coffee.

Thanks to the Fundação Araucária/CAPES for the financial support.

To everyone that may have contributed to the realization of this work supporting me, I will always be grateful.

RESUMO

Nos últimos anos, os sistemas de comunicação sem fio têm passado por um rápido crescimento, impulsionados principalmente pelos avanços de novas tecnologias como o 5G, combinadas com o progresso de dispositivos eletrônicos, que demandam cada vez um maior fluxo de dados. Embora essas tecnologias representem um avanço para os sistemas de comunicação, elas enfrentam um problema físico: a escassez de frequências do espectro eletromagnético. Devido a essa limitação, cada vez mais são necessários sistemas capazes de aperfeiçoar o uso do espectro, entregando um ambiente capaz de fornecer uma comunicação rápida, segura e eficiente onde os equipamentos sem fio possam operar. Desta forma, esta pesquisa desenvolveu um sistema de classificação automática de modulação que pode ser aplicado em ambientes de rádio cognitivo, baseado em redes neurais artificiais e implementado em uma FPGA.

A classificação de modulação é uma das tarefas executadas em um ambiente que aplica rádio cognitivo, que tem por objetivo identificar, de maneira automática, a modulação utilizada pelo sinal que chega no receptor, aplicada principalmente em ambientes em que dois usuários desejam compartilhar a mesma banda do espectro, sem que haja interferência nas suas comunicações. Esta pesquisa implementou um algoritmo de classificação automática de modulação baseado em um conjunto de dados sintético contendo cinco classes: BPSK, QPSK, 8-PSK, 16-QAM e ruído. A partir dos dados das modulações e do ruído, foram extraídos alguns parâmetros, os quais foram utilizados para treinar uma rede neural com arquitetura perceptron multi-camada, desenvolvida e treinada utilizando a API do Tensorflow/Keras. A rede neural desenvolvida foi exaustivamente testada com diferentes configurações, variando-se a quantidade de camadas, o número de neurônios, funções de ativação, entre outros parâmetros, resultando em mais de 2000 modelos possíveis, testados em mais de 200 horas. O melhor modelo destes testes foi escolhido para ser implementado em uma FPGA, demandando que ele fosse atualizado para atender às limitações do circuito. Para esta aplicação, o número de parâmetros utilizado para a classificação foi reduzido e diferentes arquitetura de redes neurais foram testadas, tendo em vista as limitações do hardware, como o limite de precisão para os cálculos e a quantidade finita de elementos lógicos disponíveis. O desenvolvimento da rede para a FPGA recorreu ao VHDL como linguagem de descrição e foi testada utilizando-se os mesmos dados da implementação no software. A implementação no hardware, entretanto, não contempla o cálculo das features, demandando que ele seja alimentado com estes dados já calculados em suas entradas.

Tanto a aplicação em software, bem como a do hardware, foram capazes de classificar corretamente aproximadamente 90% das amostras, quando o SNR era $\geq$4 dB. Entretanto, a implementação na FPGA apresentou um rápido decréscimo na sua precisão quando os níveis de SNR chegam a valores negativos, haja vista as limitações impostas nessa aplicação. Todavia, as arquiteturas implementadas nesta pesquisa superam os trabalhos similares disponíveis até o

momento, tendo em vista que utiliza um conjunto de parâmetros de entrada selecionados para a classificação que requerem menos tempo para serem processados e consomem menos recursos para a sua execução. As melhorias aplicadas na rede neural resultaram, ainda, em uma rede menor, capaz de ser implementada em uma FPGA com recursos limitados, sem que a sua precisão final fosse comprometida.

Palavras-chave: Classificação de Modulação. Redes Neurais Artificiais. VHDL.

ABSTRACT

The wireless communication systems face rapid growth over the past few years, driven by the advances in new technologies such as 5G, combined with the progress of electronic devices that demand a high data flow. Although these technologies represent a breakthrough to the telecommunication area, they face a physical limitation: the scarcity of electromagnetic spectrum frequencies. This limitation demands creating mechanisms capable of improving the spectrum utilization efficiency and delivering a fast, reliable, and secure environment in which the wireless equipment may operate. This research developed a modulation classification algorithm that shall be applied to cognitive radio environments based on neural networks and implemented in an FPGA to address this subject.

The modulation classification is one of the tasks performed in a cognitive radio environment, which aims to identify the incoming signal's modulation, primarily applied where two users want to share a frequency band without interfering in each other communication. This research implemented a modulation classification algorithm based on a synthetic dataset containing five classes: BPSK, QPSK, 8-PSK, 16-QAM, and a Noise dataset. A few parameters are extracted from these classes and then used to train a multi-layer perceptron neural network, developed and trained using the Keras/Tensorflow API. The neural network model developed was exhaustively tested with multiple configurations, varying its layers, the number of neurons, activation functions, among other parameters, resulting in more than 2000 possible models tested in more than 200 hours. The best architecture was chosen from the resulting model to be implemented to an FPGA, demanding new improvements to suit the hardware limitations. The input features used for the classification were reduced, and the network itself adapted to the hardware constraints, such as limited precision and a finite number of available logical resources. The hardware implementation used the VHDL language to its conception and was tested using the same software-based implementation data. However, the hardware implementation does not calculate the input features, requiring that the implemented neural network receive the already calculated data.

The software and the hardware-based implementations of the modulation classification achieved approximated 90% of accuracy when the SNR is equals to $\geq$4 dB. However, the hardware implementation shows a rapid decrease in its precision as the noise levels attain negative levels. Nonetheless, the architecture implemented in this research outperforms similar works developed so far, as it utilizes a set of selected input features for the classification that require less computational time and resources for its execution. The optimizations performed in the neural network architecture resulted in a tinier network, which can be implemented in a limited hardware resource FPGA without compromising the final classification capability.

Keywords: Modulation Classification. Artificial Neural Network. VHDL.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AI	Artificial Intelligence
AM	Amplitude Modulation
AMC	Automatic Modulation Classification
ANN	Artificial Neural Network
ASK	Amplitude-Shift Keying
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase-Shift Keying
CNN	Convolutional Neural Network
CPLD	Complex Programmable Logic Device
CR	Cognitive Radio
DFT	Discrete Fourier Transform
DL	Deep Learning
DNN	Deep Neural Network
DSB	Double Side Band
DT	Decision Tree
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
FSK	Frequency-Shift Keying
FSM	Finite State Machine
HDL	Hardware Description Language
HOS	High Order Statistics
LUT	Lookup Table
ML	Machine Learning
MLP	Multilayer Perceptron
MRI	Magnetic Resonance Imaging
OFDM	Orthogonal Frequency-Division Multiplexing
OTA	Over-The-Air
PAM	Pulse-Amplitude Modulation
PSK	Phase-Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RAM	Random Access Memory
ReLu	Rectified Linear Unit
ResNet	Residual Neural Network

RF	Radio Frequency
RKRL	Radio Knowledge Representation Language
ROM	Read-Only Memory
SDR	Software-Defined Radio
SNR	Signal-to-Noise Ratio
SSB	Single Side Band
SVM	Supported Vector Machine
TF	TensorFlow
USRP	Universal Software Radio Peripheral
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WandB	Weights and Biases

LIST OF SYMBOLS

$s(\cdot)$	General signal
$\alpha(\cdot)$	Signal amplitude
β	Raised cosine rolloff factor
f_c	Signal frequency
$\theta(\cdot)$	Signal phase
$(\cdot)_I$	In-phase component
$(\cdot)_Q$	Quadrature component
$\mathfrak{R}$	Real component of a complex number
$\mathfrak{I}$	Imaginary component of a complex number
$\in$	Belong
$\mathbb{R}$	Real number
T_s	Symbol time
A	Amplitude coefficient
$E[\cdot]$	Expected value
μ	Mean
$Var[\cdot]$	Variance
σ	Standard deviation
M_{pq}	High order moment
C_{pq}	Cumulant
$(\cdot)^*$	Conjugated
w	Neuron weight
b	Neuron bias
$e_k(\cdot)$	Neuron error signal
$\varepsilon(\cdot)$	Instantaneous error energy
$\varepsilon_{av}(\cdot)$	Average error energy
Δ	Correction factor of backpropagation algorithm
∂	Partial derivative
η	Learning rate
$\delta_k(\cdot)$	Local gradient
K	Bits per symbol
$\varphi(\cdot)$	Neuron activation function

CONTENTS

1 INTRODUCTION

1.1 CONTEXT AND MOTIVATION

The growth of wireless communications has led to the increasing demand for secure, reliable, and efficient communication systems in the past few decades. These structures are composed of complex equipment responsible for conditioning the information signal from the source until the final destination, combining technologies that should guarantee the signal integrity over multiple transmission scenarios. However, wireless technologies share an essential and finite resource: the electromagnetic spectrum. For that reason, the emergence of techniques that aim to improve its utilization has been gaining relevance in the past few years, especially the one known as cognitive radio.

The term "Cognitive Radio" was firstly introduced by [1] in 1999. In this work, the author described a cognitive radio environment where a few analysis is performed, and, based on the results of these analyses, the equipment used in the wireless communications are capable of adapting itself, reacting to channel changes and even predicting potential modifications that may impact the whole process. One of the analyses performed in a cognitive radio environment is the modulation classification, especially in situations where multiple users want to share the licensed spectrum band. Through this technique, a secondary user is capable of using the non-occupied band of the spectrum while the primary user is inactive; once the secondary user identifies the presence of the primary user's signal through the identification of its modulation, the secondary user can stop transmitting and start to look for a new band available.

The modulation classification and the cognitive radio technologies have the potential to alleviate the scarcity of the spectrum, and, for that reason, they are applied in a wide range of applications, both military and civilian. In military applications, these technologies can be used for spectrum surveillance, threat evaluation, or even jamming potential hostile signals [2][3]. In civilian applications, however, besides managing two or more concurrent users in a spectrum band, the emergence of new technologies has opened a new set of applications for the cognitive radio and the modulation classification. Recently, the evolution of the smart grids required the application of cognitive radio networks to support its communication infrastructure, dynamically utilizing the spectrum [4]; low power wide area networks and internet of things applications are applying cognitive radio networks to the infrastructure [5], achieving an improved spectral utilization, less transmission power constraints, increased scalability, and even longer transmission ranges. The realization of the cognitive radio, and more specifically the modulation classification for all these applications, requires the development of architectures capable of correctly handling the tasks needed, which resulted in a new subject of research over the past few years, significantly improved by the evolution of technologies such as machine learning.

In the past few years, the growth of machine learning applications results from evolving a set of technologies, such as new and more powerful hardware, bigger datasets, and new algorithms. The neural networks can learn patterns from a sufficient number of examples and then create a model that can recognize this same pattern in new data. This behavior can be exploited in many different areas, such as medicine, cybersecurity, and, obviously, telecommunications. The telecommunication applications for machine learning and neural networks cover the cognitive radio and modulation classification and is explored since the beginning of the century, where the combination of some technologies such as decisions-trees and artificial neural networks were exploited [6]. The resultant architecture improved over the years, mainly due to the emergence of more accurate machine learning algorithms, like deep learning, which now achieves high accuracy rates and can take full advantage of bigger datasets [7]. Combining these new machine learning algorithms with some frameworks such as TensorFlow, Keras, and PyTorch, accelerate the development process, and digest the complex tasks evolved to create an accurate machine learning model through accessible API functions. Nonetheless, these models may sometimes not be suitable for real-time applications such as modulation classification, as they require, in a vast majority, the execution of complex calculations. For that reason, the application of hardware-based machine learning algorithms for modulation classification has been exploited over the past few years.

The need for a quick and precise response in a device that uses modulation classification is a crucial requirement, as the minimal delay can cause the loss of important information. Allied to this, the deep learning applications implemented in a host computer limit its portability and increase the latency of moving the data from the receiver until the processor, compromising its response [8]. For that reason, hardware-based modulation classifiers, especially those implemented in FPGAs, can achieve the required time response, combined with power-efficient and high accuracy. The FPGA is capable of delivering such results mainly due to its intrinsic parallel processing architecture, where multiple information can be handled at the same time, differently from a processor, which executes the tasks sequentially. Additionally, new forms of the algorithm execution can be exploited, increasing the system's throughput, as explored by [9]. The architecture proposed by [10] implements a block topology, where each element is developed separately, optimized, and joined later to form the classification system. Further, the implementation suggested by [8] can achieve reasonable classification rates, although it requires an extensive network and combines some processing modules with the FPGA circuit.

The neural networks implemented in both the software and the hardware applications are based on calculating a set of features used as the input of each network to classify the incoming signal. These features commonly require a high amount of time to be processed in software implementation, and although it can be solved in a hardware version, it would result in high use of logical resources, for the case of an FPGA. Still, the hardware versions of neural networks often require a more extensive network to maintain accuracy, lacking a balance between the input features, the network architecture, and the system's overall performance. For that reason, this

research aimed to construct a cost-effective model of a neural network applied to the modulation classification operation and implement the resulting model in an FPGA. The constructed model had the objective of selecting the well-suitable features for the parallel processing of the FPGA, as well as evaluate the feasibility of the constructed model, in terms of logical resources usage, the response time when compared to the software implementation, and also using the fewer resources as possible, through intense testing of multiple architectures and variations of the neural network construction parameters. However, the resulting model implemented in the hardware version does not contemplate, at this point, the calculation of the features due to the time restrictions of the research. Due to this limitation, the results presented in this research can only compare the classification results of the neural networks implemented in hardware and software and not the benefits of implementing the calculation of the features in hardware, as they are not implemented in the FPGA.

1.2 OBJECTIVES

1.2.1 General Objective

This research has the general objective of constructing an cost-effective model of a neural network capable of identifying the modulation used in the incoming signal by extracting some features of it and implementing the model in an FPGA.

1.2.2 Specific objectives

This research has the following specific objectives:

1. Study of the cognitive radio and the automatic modulation classification techniques;
2. Study of the machine learning and deep learning approaches, defining the best features for the model's implementation;
3. Study of construction of reconfigurable and digital hardware implementation techniques;
4. Features calculation and selection of the ones most suitable for the implementations;
5. Neural network development and improvements;
6. Hardware-based neural network implementation and improvements;
7. Evaluation of the results and comparison between the software and hardware utilization.

1.3 DOCUMENT STRUCTURE

Chapter two provides a quick review of essential concepts required for the best comprehension of this work. It addresses the five main areas evolved: digital wireless communications, cognitive radio, statistics, machine learning, and reconfigurable hardware.

Chapter three discusses the state of the art research developed regarding modulation classification, neural networks, and applying these areas to circuits. The chapter also introduces some of the most common features and how they are combined to form a modulation classification system. It then analyzes how these parameters have been used in software applications and finally discusses the hardware applications in the area.

Chapter four addresses how this research was developed. First, the research methodology is introduced, and in a subsequent section, the implementation is presented. The chapter is organized into a discussion of each main area: the general system's framework, the neural network conception and training, and the FPGA implementation.

Chapter five introduced the results of the improved implementations, presenting the behavior of the features over different scenarios and also comparing the performance of both software and hardware implementations. Still, the chapter discusses the system's performance of each modulation recognized, and ends with a comparison between this research and the state-of-the-art similar works.

Finally, chapter six discusses the obtained results and briefly overviews this research's future works.

2 THEORETICAL BACKGROUND

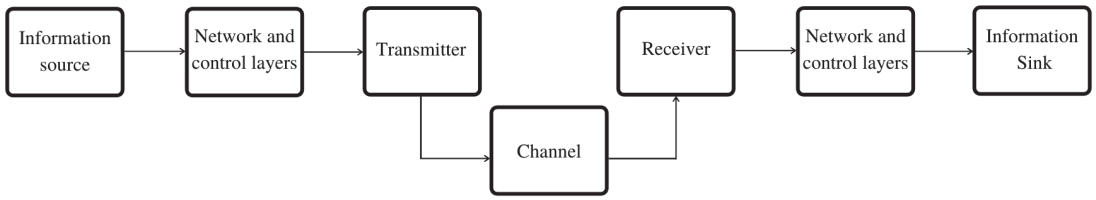
2.1 TOPICS ON DIGITAL WIRELESS COMMUNICATIONS

A communication system comprises various operations from the signal generation at a source until the final destination. These operations are performed by a set of signal processing entities responsible for coupling the signal in the best possible way for its transmission. The general setup of the communication components is shown in Figure 2.1. The source's signal is sent to a transmitter, responsible for processing the information into a convenient form for transmission over the channel. The receiver entity is in charge of converting the received signal from the channel into its original form so that the destination can understand it. Between these entities remains the network and control layers, responsible for controlling the information exchange in more complex networks that share the same physical medium, such as the Internet.

The transmitter is made up of a set of operations that includes the modulation, where the signal is appropriately impressed in a carrier signal; up-conversion stage, where the signal is converted to the radio frequency (RF) in which it will be transmitted; amplification stage, where the signal is amplified to the required power level for transmission.

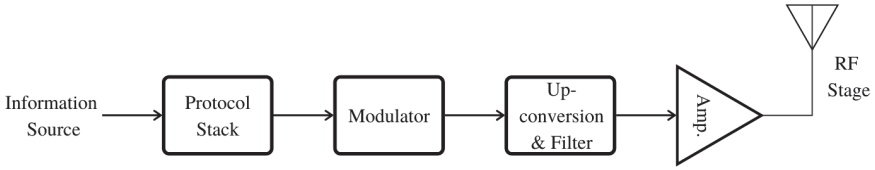
The receiver, after the amplification process through a low-noise amplifier, executes, among other attributions, the down-conversion of the received RF signal, where it is translated to a frequency where it can be easily demodulated, and the demodulation process, where the original signal is recovered. An illustration of both the transmitter and the receiver can be viewed at Figures 2.2 and 2.3. One crucial task in the communication process is *modulation*. It consists of varying the attributes of a *carrier* — that can be the amplitude, frequency, phase, or a combination of them — according to the information to be transmitted. This process is necessary for an efficient transmitting process; otherwise, non-practical antennas sizes, the impossibility of multiple signals using the same channel, or even the non-fulfillment of some design parameters would be some of the issues faced [12].

Figure 2.1: The elements of a communication system



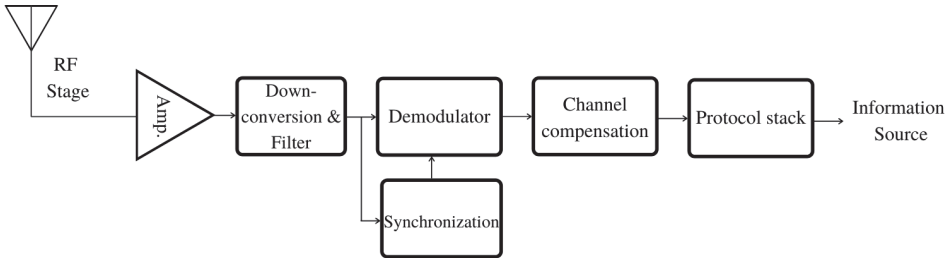
Source: Adapted from [11]

Figure 2.2: Basic radio transmitter



Source: Adapted from [11]

Figure 2.3: Basic radio receiver



Source: Adapted from [11]

2.2 COGNITIVE RADIO AND THE AUTOMATIC MODULATION CLASSIFICATION

The electromagnetic spectrum is composed of all frequencies in which licensed and non-licensed users may operate. It is made up of a combination of transmitters and receivers equipment, sharing the frequency spectrum. The problem, though, is that as a finite natural resource, it has some limitations. To better understand and overcome them, many studies have been published so far [6, 13, 14, 15, 16], pointing out many different solutions. Among them, Cognitive Radio has been gaining prominence.

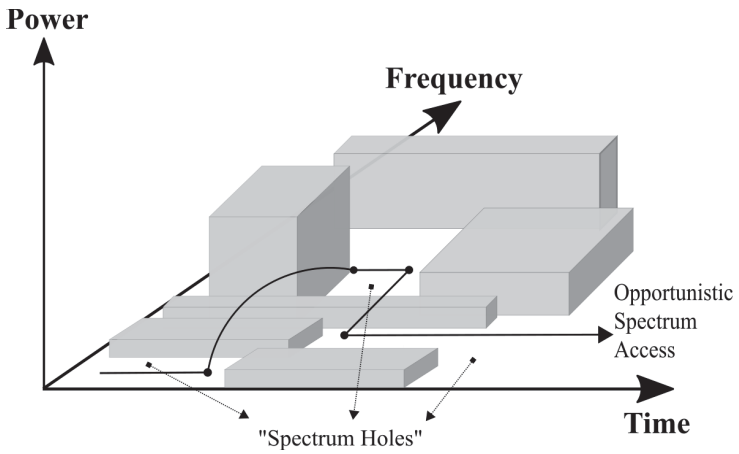
A Cognitive Radio is a system capable of performing some analysis of its surrounding, hence understanding all the conditions that may interfere with the communication process and, more importantly, reacting to them. This results in operations such as changing the carrier frequency, the transmit-power, or the modulation strategy. One of the procedures performed is based on the spectrum state take and advantage of the under-utilization of the available frequencies that might not be occupied all the time.

One of the main issues of the electromagnetic spectrum is not the scarcity of frequencies, but, in fact, the spectrum access, resulting in the concept of “Spectrum Holes”, as pointed out by [17]. In practical terms, this means that the spectrum is indeed facing an under-utilization, in which some bands are most of the time not occupied, whereas some of them are heavily used, as illustrated in Figure 2.4. That is where the Cognitive Radio takes into action, providing access to a secondary user for that band, and, consequently, making more efficient use of the spectrum, a process called opportunistic access. For doing so, the system must guarantee that once the

primary user of that band starts using it again, the secondary will stop transmitting; otherwise, it will interfere in the communication.

The opportunistic access works well in an environment where the receiver and the transmitter communicate with each other, providing a feedback channel, mainly because they share details of the communication, such as the modulation strategy adopted. Meanwhile, that is not the reality of many wireless communication systems. For that reason, the secondary user must perform what is known as *blind modulation classification*, where it should be able to identify the incoming signal's modulation without *a priori* information about it. To do so, a classifier is implemented, performing an analysis based mainly on features extracted from the signal, which include some instantaneous parameters, statistical moments, and so on. These data are then analyzed by the classifier, which can be based on decision trees (DT), supported vector machines (SVMs), or artificial neural networks (ANNs) [18]. Once the modulation is correctly identified, the secondary user can know if the incoming signal belongs to a licensed user or not.

Figure 2.4: The spectrum hole concept



Source: Adapted from [19]

The Cognitive Radio and the Automatic Modulation Classification are an important part of the work constructed in this research. However, although it represents an important motivation for this research, the modulation classification for Cognitive Radios, especially applied to the spectrum hole concept, is not the only application and motivation for its development. The correct classification of an incoming signal may be an important factor in interference monitoring or even regulatory and defense applications [7]. For that reason, these situations will be more deeply investigated in the State of the Art chapter, where some of the more recent researches that implement CR similar works are discussed.

2.3 ARTIFICIAL NEURAL NETWORKS AND MACHINE LEARNING

In recent decades, one technology that has been gaining prominence is Artificial Intelligence (AI) due to its capacity of handling vast amounts of data, reconfigurability, and ability

to solve problems that would be intellectually difficult for humans. Combining this potential with the fast advances that enables the AI applications, both in software and in hardware, two new fields emerged inside the AI: Machine Learning and Deep Learning.

Responsible for the most basic AI tasks, Machine Learning (ML) is the field in charge of constructing algorithms capable of analyzing raw data and extracting useful information from them. The extracted information is based mainly on pattern recognition, where the machine is "trained" using a vast amount of data, from where its algorithms then learn how to perform the tasks for which they were designed [20]. Although capable of performing essential tasks, ML systems have limitations that prevent their application in real-world problems. That occurs because they are heavily based on how the data is introduced to the system, determining its success or failure based upon a judgment about what may be relevant for classification or not, performed by the one training the system. For that reason, a more developed system was needed, and researchers started to work on what we currently know as *deep learning*.

Deep Learning (DL) can be understood as a more complex technique to develop AI applications that are not entirely resolved by ML systems. DL is located inside ML and is intended to work together with this technology, solving its limitations. The better outcomes accomplished in DL algorithms result from improvements regarding how the data is analyzed, combined with computational advances or improved models. One of the breakthroughs responsible for DL algorithms' growth is representing complex information with simpler small pieces of data. Instead of using only relevant information provided by an operator, DL can work with small portions of data that in the first look may seem irrelevant, but when combined, are capable of providing relevant insights. This kind of approach is based on the biological brain, and that is why DL is also known as the Artificial Neural Network.

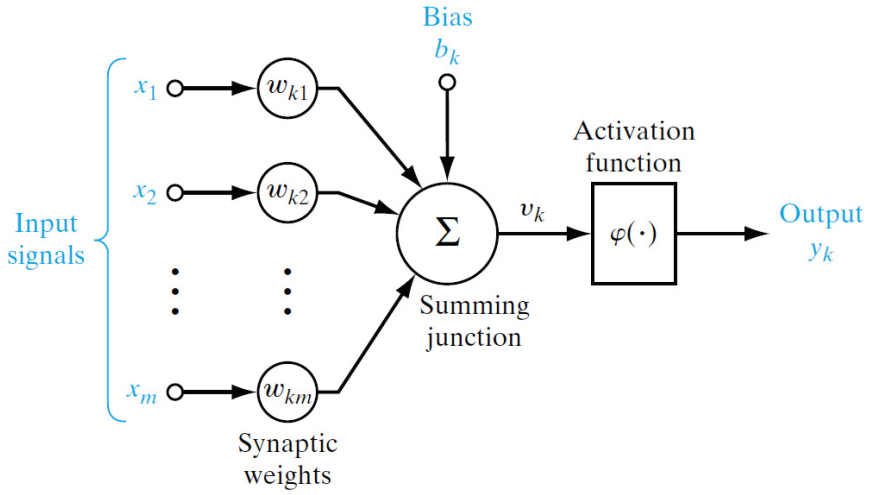
The brain is made up of a vast organization of tiny cells, known as *neurons*. These cells act like small processing units, capable of working with small stimuli provided by the environment and transmitting the acquired knowledge to each other through a process called by *synapses*. The combination of thousands of these cells is responsible for performing various computations such as pattern recognition, motion control, environment perception, and so on [21]. Like these cells represent the most basic unit in a biological brain, it does in ANN, where an artificial neuron is created to act like the neurons present in the brain, each processing small pieces of data and working together to perform a specific task. In Figure 2.5, a model of an artificial neuron is presented, where some essential elements can be identified: [21]:

1. A set of synapses, each one characterized by a weight. In the representation, a signal x_j at the input of synapse j connected to a neuron k is multiplied by the synaptic weight w_{kj};

2. An adder, responsible for summing the input signals, weighted by the respective synaptic strengths of the neuron;

3. An activation function for limiting the amplitude of the output of a neuron.

Figure 2.5: Nonlinear model of a neuron



Source: [21]

Thus, the neuron can be mathematically described as [21]:

$$v_k = \left(\sum_{j=1}^{m} w_{kj} x_j \right) + b_k, \tag{2.1}$$

and

$$y_k = \varphi(v_k), \tag{2.2}$$

Where $x_1, x_2, \cdots, x_m$ are the input signals; $w_{k1}, w_{k2}, \cdots, w_{km}$ are the respective weights of the neuron k; b_k is the bias; $\varphi(\cdot)$ is the activation function and y_k is the output signal of the neuron.

The bias b_k present in the neuron is responsible for lowering or increasing the activation function's input, acting like an affine transformation [21], to fit the data for a better prediction. The weights, in turn, are in charge of storing the acquired knowledge through different connection strengths. At the end of the neuron, the activation function defines its output. Depending on which application the DL system is being designed for, various functions may be applied, being the sigmoid the most used.

The *sigmoid* function is one of the most common activation functions used in ANN. One example is the logistic function, defined by [21]:

$$\varphi(v) = \frac{1}{1 + \exp(-av)}, \tag{2.3}$$

Where a is the slope parameter. This function assumes a continuous range of numbers between 0 and 1 and is differentiable, an essential feature for neural networks. An illustration of the sigmoid function's behavior with $a = 1$ is presented in Figure 2.6(a).

In some cases, it is desirable for the range of the function to vary from -1 to 1. For this case, the corresponding form of a sigmoid function, the *hyperbolic tangent function*, is commonly used. It is defined as [21]:

$$\varphi(v) = \tanh(v). \tag{2.4}$$

The behavior of the tanh function is illustrated in Figure 2.6(b).

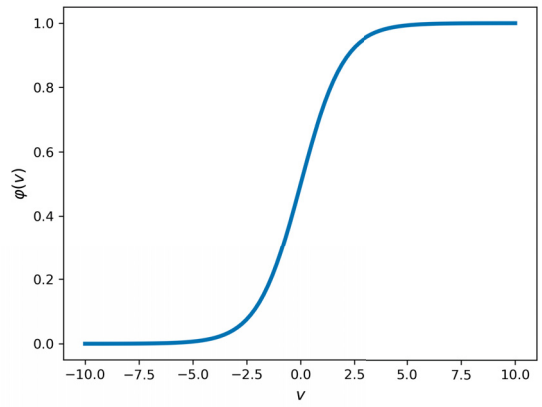
Another widely used activation function is the *rectified linear unit* (ReLu), whose primary behavior is to return the input value directly or 0 if the input value is 0 or less. It may be used in situations where the sigmoid and tanh functions' saturation behavior should be avoided. The ReLu function is defined by [22]:

$$\varphi(v) = \max(0, wv + b), \tag{2.5}$$

Where w is a weight vector and b is a bias. An illustration of the ReLu behavior with $w = 1$ and $b = 0$ is shown at Figure 2.6(c).

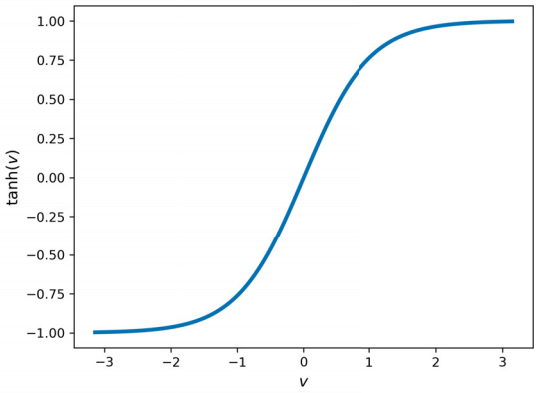
Figure 2.6: Activation functions behavior

(a) The sigmoid function



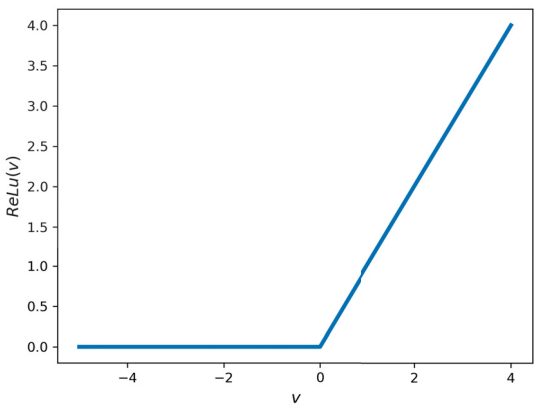
Source: The author (2020)

(b) The tanh function



Source: The author (2020)

(c) The ReLu function



Source: The author (2020)

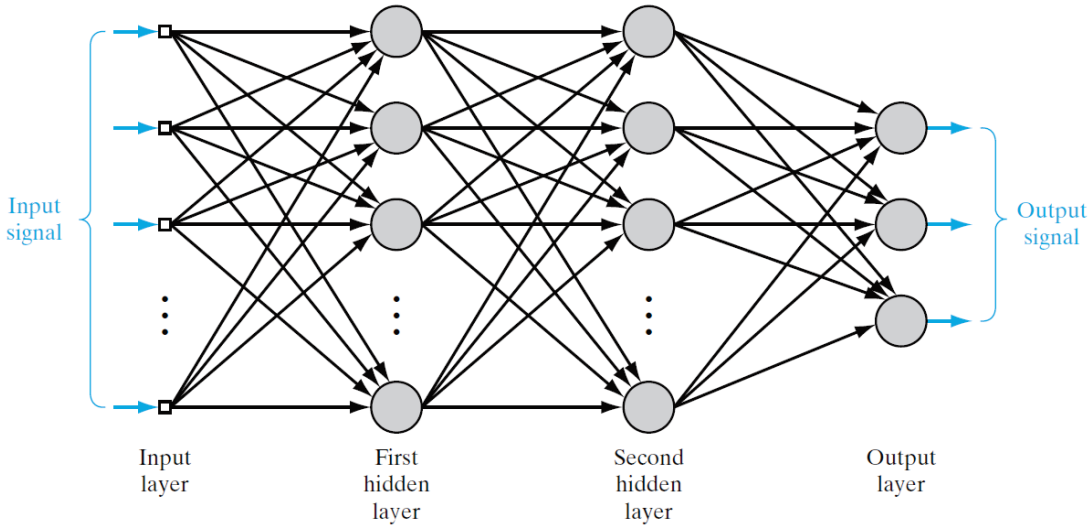
The last activation function is the *softmax*. This function's output can be interpreted as a probability distribution because it is always positive and sum up to 1. Applied to the output of a neural network, it can convert the usual real numbers provided by the network to probabilities, enabling the interpretation of the estimated probability that the answer is correct. It is defined by [22]:

$$\varphi(v)_i = \frac{e^{v_i}}{\sum_{j=1}^{K} e^{v_j}}, \tag{2.6}$$

Where v is the input vector, v_i are all ith values of the input vector v and K is the number of classes of the classification output.

For the construction of an ANN, different architectures are available. The choice of the right one depends on the task that is willing to apply the network, as some are well-suited for specific tasks. A widely used architecture is the *multilayer perceptron* (MLP), commonly used in classification activities. In this arrangement, several units of *perceptrons* - single neurons, capable only of classifying linearly separable patterns - are combined in one or multiple layers. These layers are known as *hidden layers*, as their data is not directly visible by the input and output layers. The number of neurons in each layer and the number of layers is arbitrary, being necessary to evaluate the entire network for each application to decide the number of neurons and layers capable of delivering the best results. At these layers, each neuron is connected to all neurons in the previous layers, a characteristic which is known as *fully connected* MLP. An illustration of an MLP with two hidden layers is shown in Figure 2.7.

Figure 2.7: The MLP architecture



Source: [21]

The neurons of the ANN need to learn about the tasks they will execute. This process can be made using two approaches: *unsupervised* and *supervised* learning, being the supervised method applied to this research. Through this learning process, the ANN is provided with a set of input-output examples, which contains the desired response and with the input parameters that

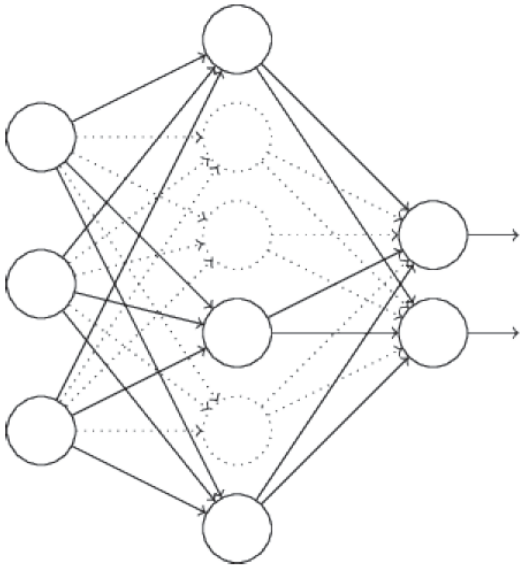
should result in that response. With these criteria, the network is adjusted until its output reaches the goal. Once this process is concluded, the knowledge is stored at each neuron's synaptic weights that make up the network. The learning process is executed following an algorithm of continuous optimization of each neuron's weights, known as *back-propagation algorithm*, where each value is updated continously until it converges to the slightest error possible. The described method of learning is also known as *batch learning* [21] and it consists of adjusting the synaptic weights only after all the N examples of a training sample are passed through the neural network, forward and backward, which constitutes one *epoch* of training.

The learning process's objective is to develop a network model capable of dealing correctly with data never used in creating or training the network. When this is achieved, it is said that the ANN is capable of *generalize* for any data [21]. However, in the course of training, one thing that may occur is memorizing the training data by the network. It memorizes the data examples due to some feature present in the training data, but that is not present in the actual scenario that the network is being trained to operate. In such a case, the network fails in creating a generalized model, and it is said that the network is *overfitting* [21]. To solve that, some techniques can be applied. The simplest one is adding a new layer between the network's hidden layers to temporarily modify the learning process, known as *dropout* layer [22]. The dropout layer is responsible for deleting the connection between some neurons from one hidden layer to another, thus simulating the creation of multiple diverse network setups in each epoch of the training process, as presented in Figure 2.8. After one modification is completed, a new set of random neurons is chosen to be temporarily deleted, and this process continues through all the training operations. In the end, the dropout results in averaging the overfitting of multiple simulated networks, and, as each network overfits differently, the net effect of the dropout layer is to reduce the overfitting phenomenon significantly. Once the network model is properly trained, its effectiveness in classifying new data samples should be evaluated. To do so, some metrics of the model behavior are analyzed, including the *accuracy, precision, recall, and the* f_1 *score*. These parameters are capable of summarizing the model behavior, based on four categories [23]: correctly classified samples that belongs to the positive class or *true positives* (TP); correctly classified samples that belongs to the negative class or *true negative* (TN); incorrect positive prediction or *false positive* (FP) and the incorrect negative prediction or *false negative* (FN). These categories are arranged in a table known as *confusion matrix*, which provides visual information about an ANN model's behavior. An example of a confusion matrix is shown in Figure 2.9.

The first metric is accuracy. It relates the number of correct predictions with the number of all samples tested [23]:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.7}$$

Figure 2.8: The dropout setup



Source: [22]

The second one is precision, which is capable of measuring how many of the samples classified as positive are actually positive [23]:

$$precision = \frac{TP}{TP + FP}. \tag{2.8}$$

The third one, the recall, measures how many of the positive samples are captured by the positive predictions [23]:

$$recall = \frac{TP}{TP + FN}. \tag{2.9}$$

The last one, the f_1-score, is given by the harmonic mean between the precision and the recall and is widely used in binary classification with imbalanced datasets. It is defined by [23]:

$$f_1 - score = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{2.10}$$

2.4 RECONFIGURABLE HARDWARE COMPUTING

Introduced by Xilinx in the 1980s, FPGAs or *Field Programmable Gate Arrays* represents an advancement to its predecessors, the CPLDs (Complex Programmable Logic Devices), devices capable of implementing programmable circuits but with limitations regarding the complexity of functions that can be implemented. An FPGA implements some changes in architecture, size, performance, coast, and construction technology, to name a few differences responsible for its continuous growth since its creation.

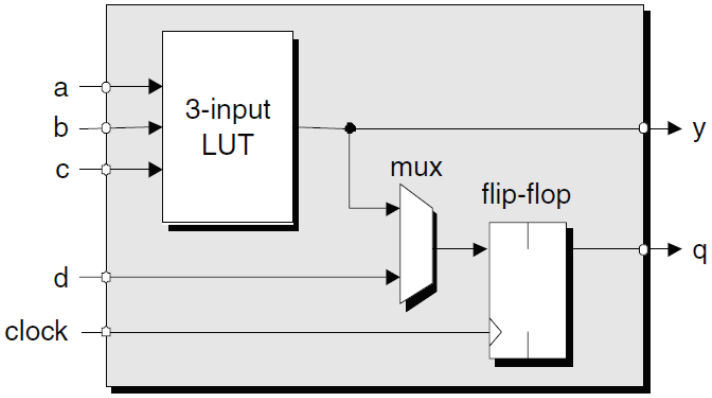
Figure 2.9: Confusion matrix

		Predicted Classes
True Classes	True Negative	False Positive
	False Negative	True Positive

Source: The author (2020)

An FPGA is made of a matrix of combinational logic blocks, where each one of these blocks contains mainly three key elements: a lookup table (LUT), a register that can act as a flip-flop or a latch, and a multiplexer [24], as seen in the Figure 2.10. The combination of hundreds

Figure 2.10: The logical block of a FPGA device



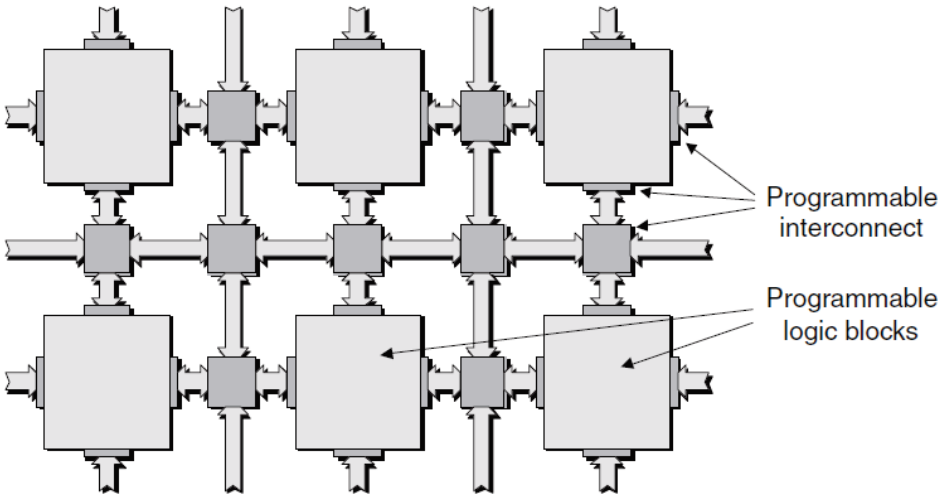
Source: [24]

of thousands of these logical blocks makes up an FPGA, as illustrated in Figure 2.11. As shown in the Figure, every block and every interconnection between them can be programmed, resulting in various combinations. The flexibility provided by these interconnections makes it possible to design hardware well-suited for a specific algorithm, thus constructing only the operations needed to execute it, in contrast to a typical processor that must cover all the instructions possible. Additionally, to these logic blocks, the device may include I/O pins and pads responsible for connecting the circuit to the external world.

To construct and implement a circuit inside an FPGA, a *Hardware Description Language* (HDL) may be used. One of them is the *Very High-Speed Integrated Circuit Hardware Description Language* (VHSIC HDL or simply VHDL), introduced by the United States Department of Defense

in 1980. This language's objective was to standardize how integrated circuits were generated and solve issues such as the lack of standardized documentation or even the incompatible simulation and design tools used so far. By the end of the decade, the VHDL became an IEEE(IEEE 1076) standard, with its first release in 1993. Since this date, some other versions have been released, and VHDL figures as one of the industry-standard languages for the circuit conception.

Figure 2.11: A simple FPGA architecture



Source: [24]

3 STATE OF THE ART

The emergency and improvement of technologies such as neural networks and reconfigurable digital hardware in microelectronics, combined with a vast demand for increasingly efficient communication systems, have opened a new research field that combines both subjects. Early in the 1990s, many researchers started to study how these technologies could be merged to provide fast, accurate, and reliable systems to develop a complete cognitive radio environment[1, 6, 13, 17, 25]. This chapter presents a review of state-of-the-art cognitive radio systems, introducing some fundamental concepts developed regarding the technology, applications, and associated drawbacks.

3.1 THE COGNITIVE RADIO

Cognitive Radio deals with the fact that the electromagnetic spectrum is a limited resource. The spectrum unlicensed bands' availability is scarce, although the frequency spectrum can be under-utilized for a given location. Thus, the CR can provide reliable communication and efficient use of this resource. Although its use has been driven by advances in a combination of technologies in the past decades, the term *Cognitive Radio* was firstly introduced in 1999 by [1]. In this work, the author introduces the *Radio Knowledge Representation Language* (RKRL), a new approach in which the Radio architecture is capable of exchanging information between the RF network structure, hence adapting itself accordingly to the environment changes, the user needs, and even anticipating potential changes. Although it represents only the initial studies in Cognitive Radio, as pointed by the author, this study represents the beginning of a set of field-related research regarding RF engineering. Inspired by this study, a few years later, in 2005, [17] deepened the concept of Cognitive Radio, defining the *Cognitive Cycle*, a set of tasks performed by CR in terms of signal processing and machine learning techniques to gather all the information required for these tasks, such as interference estimation, spectrum holes, or the estimation of the required transmit-power. As pointed by the author, dynamic spectrum management, one of the tasks performed by the Cognitive Cycle, consists of analyzing the spectrum state, looking for opportunistic accesses — the spectrum holes, making sure that a reliable communication process is permanently established. To do that, one technique introduced is the change of the modulation strategy adopted, according to the time-varying conditions of the environment, choosing the right one based on the performed analysis. However, according to the author, this structure demands a feedback channel between the transmitter and the receiver, a condition that is not always suitable in real-world applications.

The use of the Cognitive Radio capabilities have been expanding over the past years. Besides its application in the spectrum utilization improvements when analyzing the spectrum holes, other applications have been exploited by researchers, such as the work introduced by [7].

According to the authors, the process of correctly identifying the modulation of an incoming RF signal represents a key factor in areas such as interference monitoring, radio fault detection and even regulatory and defense applications. Also, the correct identification of the signal have important applications in radio sensing and communication systems, as pointed by them.

The CR's spectrum sensing capability is a possible solution for spectrum management and other situations where the modulation classification may be applied. In this way, two main strategies can be adopted: creating a feedback channel between the transmitter and the receiver equipment or the application of modulation classification without feedback, where the modulation scheme is recognized based on signal processing tasks performed in the received data. Although both of these possible solutions may be capable of eliminating the possible mismatches between the receiver and the transmitter equipment setups due to the ideally real-time configurations performed in the CR environment, creating a feedback channel is not always suitable in real-world applications. For that reason, this research focuses on developing a modulation classification algorithm that has no dependencies on the configurations of the transmitted signal, being able to process the received data and extract all the information required. This kind of operation is well suited for spectrum sensing, precisely the modulation classification task. This enables its application in an environment where a secondary user uses a spectrum hole, where the primary user has stopped its transmission for a while; once it starts transmitting again, hence occupying its spectrum band that corresponds to the spectrum hole temporarily occupied by the secondary user, it should be able to detect the presence of the original signal —identifying its modulation and, therefore, releasing the frequency occupied and stopping its transmission. Additionally, as pointed previously, another possible application may also include the interference monitoring or even regulatory and defense utilizations.

3.2 MODULATION FEATURES

The modulation classification operation is formed by two main steps: calculating features from the incoming signal and decision making based on the extracted data. The features step calculation is based on exploring the signal behavior on three fronts: phase, amplitude, and frequency. Combined with these signal characteristics, some other statistical data can also be used for signal classification.

The work proposed by [13, 6, 25] introduces some essential features based on the signal's characteristics. The first feature proposed is σ_{aa}, the standard deviation of the absolute value of the normalized-centered instantaneous amplitude of the input signal [25]:

$$\sigma_{aa} = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} A_{cn}^2(i)\right) - \left(\frac{1}{N}\sum_{i=1}^{N} |A_{cn}(i)|\right)^2}, \tag{3.1}$$

Where N is the number of samples of the intercepted signal frame and the $A_{cn}(i)$ is the value of the normalized-centered instantaneous amplitude at time instances $t = i/f_s (i = 1, 2, \ldots, N)$, defined by:

$$A_{cn}(i) = A_n(i) - 1, \tag{3.2}$$

Where

$$A_n(i) = \frac{A(i)}{m_a}, \tag{3.3}$$

Where m_a is the average value of one frame:

$$m_a = \frac{1}{N} \sum_{i=1}^{N} A(i). \tag{3.4}$$

The second proposed feature proposed by the authors is σ_{fa}, the standard deviation of the absolute value of the normalized centered instantaneous frequency of non-weak samples [25]:

$$\sigma_{fa} = \sqrt{\frac{1}{N_c} \left(\sum_{A_n(i) > a_t} f_N^2(i) \right) - \left(\frac{1}{N_c} \sum_{A_n(i) > a_t} |f_N(i)| \right)^2}, \tag{3.5}$$

Where N_c is the number of samples where $A_n(i) > a_t$, being a_t a threshold value that selects all the non-weak samples, because bellow of this value the signal is very sensitive to noise; $f_N(i) = f_c(i)/r_b$, being r_b the symbol rate of the signal, $f_c(i) = f(i) - m_f$ and $m_f = 1/N \sum_{i=1}^{N} f(i)$.

The third features proposed is the kurtosis of the normalized instantaneous frequency, μ_{42}^f, defined by [6]:

$$\mu_{42}^f = \frac{E\left\{f_n^4(t)\right\}}{\left\{E\left\{f_n^2(t)\right\}\right\}^2}, \tag{3.6}$$

Where $f_n(t)$ is the normalized instantaneous frequency, determined by $f_n(t) = f(t)/\max f(t)$, where $f(t)$ is the signal's instantaneous frequency.

Along with those features, the authors proposed some other parameters like the power spectral density of the signal's amplitude, the standard deviation of non-weak instantaneous and absolute values of the phase of the signal, the kurtosis of the amplitude, and even the signal spectrum symmetry around the carrier frequency and is characterized.

In [26], the authors introduced a few statistical features responsible for providing a low-complexity and robust method for modulation classification. As pointed by the authors, the High Order Statistics (HOS) is chosen due to its robustness against Gaussian noise, constellation rotation, phase jitter, and the characterization of the shape of the distribution of the noisy

constellation. The HOS introduced by [26] are up to fourth-order and are based in the High Order Moments, from where the cumulants are obtained and defined as:

$$C_{40} = M_{40} - 3M_{20}^2, \tag{3.7}$$

$$C_{42} = M_{42} - \left|M_{20}^2\right| - 2M_{21}^2. \tag{3.8}$$

Additionally to their robustness against the noise effects, these features are adopted because they are computationally less expensive than the ones presented by [6], which requires, for example, the application of a Discrete Fourier Transform (DFT).

The work proposed by [27] also uses the HOS as the input features for a modulation classification system. However, it utilizes an even higher-order cumulants, such as C_{60}, C_{61}, C_{62}, and C_{63}, being the last one expressed as:

$$C_{63} = M_{63} - 9M_{21}M_{42} + 12M_{21}^3 - 3M_{20}M_43 - 3M_{22}M_{41} + 18M_{20}M_{21}M_{22}. \tag{3.9}$$

In work proposed by [28], the authors implement a combination of all the previously presented features with some new ones, such as the skewness of the signal distribution, peak-to-rms and peak-to-average ratios, the mean value of the signal's instantaneous amplitude, normalized square root value of the sum of amplitude signals, and the standard deviation of the normalized signal's amplitude. According to the authors, these features were chosen based on how good was the separation between each other in the histogram of all features versus each modulation.

Combined with the previous features, [29] also examines a new feature that results from a combination of HOS features, that the author calls of v_{20}. Additionally, another feature proposed by him is the signal's power ratio, β.

3.3 SOFTWARE-BASED AMC'S ARCHITECTURES

The implementation of a modulation classification algorithm in software can be based on a few different architectures. The most common are the decision-theoretic or decision-tree and the one based neural networks solutions, each with advantages and drawbacks.

In the work proposed by [6, 13, 25] these two approaches are introduced. The decision-theoretic method uses the features' resulting values to separate the modulations based on a threshold value, which is chosen based on the probability of correct decisions, performed on 400 realizations of each modulation of interest, at 15 and 20 dB of signal-to-noise ratio (SNR) [13, 25]. The final classification in this method is based on the majority rule, in which the modulation with the most significant number of repetitions in a test is chosen. However, the method based on an artificial neural network can learn each modulation's feature values and label the received signal with the most probable modulation. The training process is based on 50 frames with 2048 samples at SNR 15 and 20 dB of each modulation analyzed [6]. Both methods are evaluated

using 400 frames, also at 15 and 20 dB of SNR. The algorithms can classify both analog and digital modulations (AM, DSB, VSB, LSB, USB, FM, 2-ASK, 4-ASK, 2-PSK, 4-PSK, 2-FSK and 4-FSK), where the decision-theoretic algorithm achieves 94% of success rate at SNR 15 dB; meanwhile, the ANN achieves 96%, at the same noise level. The DT architecture proposed is also known as *maximum-likelihood classifier* [30], and, although it is one of the most common approaches adopted regarding the AMC matter and accomplishes good results at a relatively high SNR and when channel model and channel parameters are well known, it has some drawbacks, such as the classification based on estimated thresholds, which may not be precise enough to represent the actual signal accurately and the computationally expensive and high time required to calculate the signal features, which may prevent its application in real-time scenarios. These issues motivated a few studies to decrease the whole process complexity, primarily focusing on the signal's parameter extraction.

In the article introduced by [26], the authors used a DT-based approach for classification, with thresholds based on theoretical values for multiple constellation types, established from the ensemble average of an ideal noise-free constellation, in a constraint of unity energy. To prove the robustness of the developed method, the authors realized 1000 classifications for each possible result, separated into two modulation groups: the first one with BPSK, 4-PAM, 4-QAM, and 8-PSK; the second with BPSK, PAM, 4-PSK, 8-PSK, V32, V29, V29c, and 4-QAM. These two groups were submitted to a few testes, where the impact of various signal impairments was evaluated, such as the presence of Additive White Gaussian Noise (AWGN), non-Gaussian noise, or phase and frequency offsets. The first group containing the four modulations achieves an error-free performance when the SNR is about 8 dB in an AWGN channel, where the sample size of each modulation is at least 250. When there is a phase offset present, the system can correctly classify around 99% of the input samples, with an SNR of 12 dB and a sample size of 200. In contrast, for the eight-modulation group, the authors' results are under an SNR of 20 dB and with 500 samples at least, achieving around 97% of accuracy, with no information about the impairments applied. Although requiring a relatively high SNR level to provide accurate results, this article can demonstrate the robustness of the HOS parameters against the most common channel impairments, responsible for the majority of the mismatch classifications in AMC systems.

The relatively high SNR levels required for an error-free classification in the architecture presented by [26] results in a disadvantage for a reliable AMC. As the modulation classification executes a crucial step in the CR environment, the classification algorithm must provide a fast and accurate response; otherwise, the whole communication process is affected. Many studies have been published so far to develop a more accurate classifier, combining some of the features already presented or making adjustments in the data-processing algorithm [26, 6, 29]. Nonetheless, even though some works already use neural networks combined with modulation classification [13], it was just at the beginning of the last decade that this technology thrives as an essential tool that could enhance the AMC models.

In the work introduced by [29], the authors describe an AMC model based on features extraction that adopts an ANN as the classification method. In this article, the authors implement their classification model in three main blocks: the first one is responsible for extracting the features from the input signal that will be used for the modulation recognition; the second executes the neural network training, and the third evaluates the model created in the previous step. The feature extraction process executed in the preprocessing step combines features based on the amplitude, phase, and frequency of the signal with HOS features. The ANN architecture for the classifier is based on an MLP, with an input layer with eight neurons —value that corresponds to the number of features, a hidden layer with 15 neurons and one output layer, with 13 neurons, that corresponds to 12 possible modulations and a noise signal. The system is capable of handling the following modulation schemes: 2-ASK, 4-ASK, 2-FSK, BPSK, QPSK, AM, DSB, SSB, FM, OFDM, 16-QAM, and 64-QAM. The data utilized on the AMC model's second and third steps were generated utilizing the MATLAB platform, with 60000 examples. These samples were degraded using an AWGN channel model, with the SNR values of -5, 0, 5, 10, 15, and 20 dB. According to the authors, the developed classifier can recognize above 99% of the input samples for signals with SNR values from 0 dB and upward. However, the authors claim that even at -5 dB of SNR, the system recognized above 95% of the samples.

The ML-based methods for modulation classification are capable of delivering good outcomes, even at low SNR levels. However, machine learning systems are highly dependent on how the data is introduced to the training phase and not capable of adequately dealing with multi-dimensional datasets. These characteristics, combined with the last few years' increase of hardware evolution, mainly graphic cards capable of executing ML-focused operations, culminated in the emergence of deep-learning-based methods for modulation classification[28].

In the article introduced by [28], a Deep Neural Network (DNN) is proposed for modulation recognition. The model can recognize five modulation schemes: BPSK, QPSK, 8-PSK, 16-QAM, and 64-QAM, degraded by AWGN with SNR values of -5, 0, 5, 10, and 15 dB and Rician fading with Doppler values of 100Hz and 300Hz, with no information provided about how the simulation data was generated. The classification relies on 21 features extracted from the input signal. The proposed DNN architecture has five layers: one input layer with 21 neurons; three hidden layers, with 500, 200, and 40 respectively; and the output layer, with 5 neurons. The DNN was tested over 20000 samples and, according to the article, an error-free classification in both AWGN and Rician fading is accomplished at SNR equals to 10 dB, although in -5 dB of SNR level, the DNN proposed is capable of recognizing >99% of the input samples, also in both AWGN and Rician fading.

The advantages of Deep Learning algorithms are also explored in the work proposed by [7], where three main approaches for modulation classification are examined: a baseline method, that consists of an improved DT-based classification; a Convolutional Neural Network

(CNN) [1] with seven hidden convolutional layers combined with a Max Pooling layer between them and a Residual Neural Network (ResNet) [2], with six residual stacks layers. To explore these architectures, the authors created two datasets: one containing only synthetic data and the second with Over-The-Air (OTA) measurements, both with a series of impairments applied, such as AWGN and Rayleigh fading. These datasets are made up of 24 modulations, which were grouped into two sets: one containing modulations with low information density, representing impaired environments — classified by the authors as "Normal" — and the second with high order modulations, usually applied in real-world scenarios — denoted as "Difficult", as illustrated at table 3.1, from where the HOS features are extracted.

Table 3.1: Modulation classes

Normal Classes	**Difficult Classes**
OOK, 4-ASK, BPSK, QPSK, 8-PSK, 16-QAM, AM-SSB-SC, AM-DSB-SC, FM, GMSK, and OQPSK	OOK, 4-ASK, 8-ASK, BPSK, QPSK, 8-PSK, 16-PSK, 32-PSK, 16-APSK, 32-APSK, 64-APSK, 128-APSK, 16-QAM, 32-QAM, 64-QAM, 128-QAM, 256-QAM, AM-SSB-WC, AM-SSB-SC, AM-DSB-WC, AM-DSB-SC, FM, GMSK, and OQPSK

Source: The author (2021)

In each sample of the synthetic dataset, the impairment values were randomly generated, guarantying a new uncorrelated random channel initialization. The frame length is 1024 samples long, with an SNR varying from -20 to 30 dB. To best evaluate the three proposed classification methods, the authors investigated the model's behavior under a few different situations, such as only AWGN degradation, varying the NN's depth, or different modulation types.

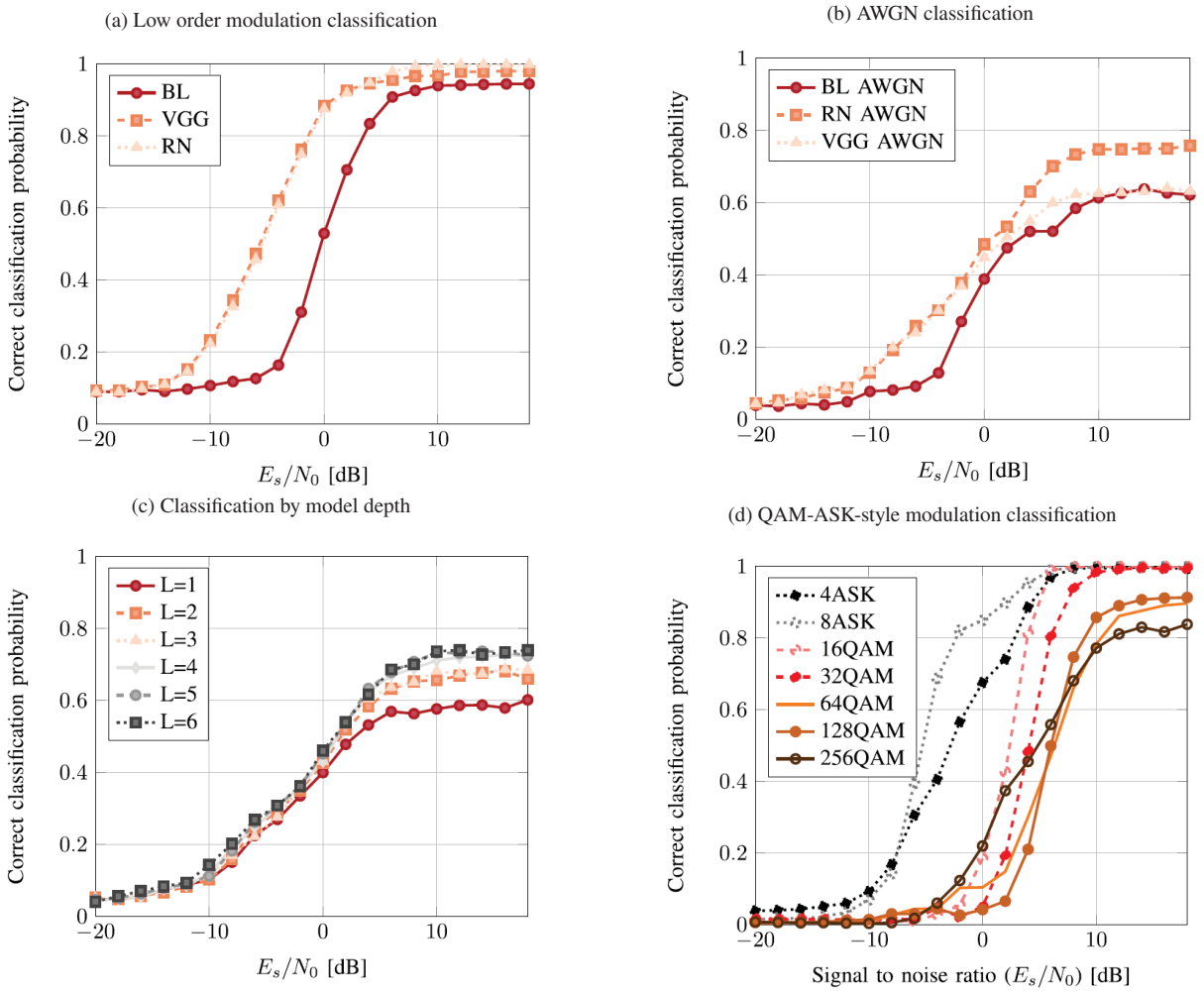
For the low order modulation types, the three methods are capable of classifying at least 90% of the samples correctly with an SNR level of ≈5 dB, as illustrated in Figure 3.1(a). This result from the training of over one million examples and in AWGN conditions. In this situation, it is clear that ResNet and the CNN (Referred to as VGG due to the architecture adaption implement by the authors) perform with roughly 5 dB higher sensitivity than the baseline method. When it comes to the full dataset impaired by the AWGN model, the classification methods were trained with approximately 240k examples, with the best results accomplished by the ResNet, with ≈80% of correct classifications with 10 dB of SNR, as presented a Figure 3.1(b). The authors also demonstrated that the deeper NN, the more it correctly classifies the samples. Figure 3.1(c) shows that the greater the number of residual stacks at the ResNet, the higher becomes its correct classification probability. Another situation analyzed by the authors is the different modulations

[1] A CNN is one class of Neural Networks that is usually applied to images recognition and that is based in a different configuration when compared to an MLP, because its input usually requires none or minimal preprocessing of data, being the NN able to find patterns by itself.

[2] A ResNet is a class of Neural Networks with an architecture similar of an MLP, excepts that it skips some connections between its neurons to avoid problems in the learning process.

schemes' response to the classification methods. As reported by them, all modulations achieve 80%+ of accuracy when SNR is about 10 dB. One example for QAM-ASK-style modulations is shown at Figure 3.1(d). According to the authors, this work also implements a synthetically trained model applied to an OTA dataset, where a loss of around 7% of accuracy is faced, demonstrating the possibility of applying a channel simulated environment AMC in real-world applications.

Figure 3.1: Response of the methods under different situations



Source: [7]

Legend: 3.1(a) represents the low order modulation classification comparisson between the three NN methods proposed by the authors, where BL stands for BaseLine method, VGG is the VGGNet, an architecture of CNN and RN stands for ResNet. 3.1(c) compares the classification by the ResNet depth, where L represents the number of residual stacks.

3.4 HARDWARE-BASED AMC'S IMPLEMENTATIONS

The construction of NN-based systems for modulation classification is, in a vast majority, based on software tools. This strategy's choice remains in the fact that software applications are

easy to develop and maintain compared to dedicated circuits. Additionally, they can reduce the time-to-market of a final solution, and over the years, a few libraries for both signal processing and machine learning applications tend to facilitate the construction of such systems. Although apparently advantageous, one of the main drawbacks of software-based applications is the time required for performing some operations, especially those that demand complex calculus. This represents a significant drawback in systems requiring almost real-time analysis and response, such as AMC and CR.

For that reason, over the years, researchers have developed some strategies for implementing neural networks in dedicated circuits, especially in FPGAs, due to its intrinsic parallelism and reconfigurability, which match those required by NN and also because of the unique performance provided by FPGA circuits. By the end of 1990, [31], and [9] proposed one of the first NN architectures implemented in FPGAs. In both works, the authors proposed implementing an MLP based on dynamic reconfiguration hardware. The weights were obtained using computer simulations and subsequently described using VHDL. Specifically, [9] discuss three new forms of parallelism that shall be explored: *spatial*, where every neuron in the same layer runs simultaneously, *algorithmic*, concerning the execution algorithm itself and *pipeline* execution through the layers for higher throughput. The neuron proposed by [31] is based in an architecture similar to the one presented by the Figure 2.5 and implements the sigmoid (2.3) as the activation function, based in a LUT. Each neuron's single unit performs the required computation with the previously saved weights and stores the result in a Random Access Memory (RAM) instance. The results are then passed through the sigmoid LUT and a final multiplier that gives the final result. Even though none of the works properly discuss their results, they provide the architecture foundations in which the further FPGA-based NN were implemented.

The architecture proposed by [32] implements a top-down methodology in which each component is implemented separately and joined in the end. All of them are constructed based upon the parametric VHDL, using the generic option, and using a model-based mainly in:

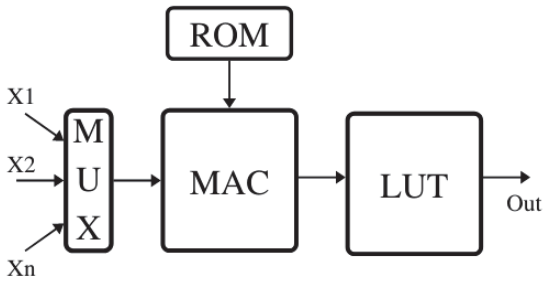
- A memory circuit where the synaptic weights are stored;
- A Multiply-Accumulate (MAC) unity which computes the weighted sum;
- Lookup table for the sigmoid function.

When all of these elements are disposed of together, they form the basic units of the ANN — the neuron and the layers — and the junction of these units form the top-level elements of the proposed architecture modular structure. The representation of the proposed neuron and a simple layer is presented at Figures 3.2(a) and 3.2(b).

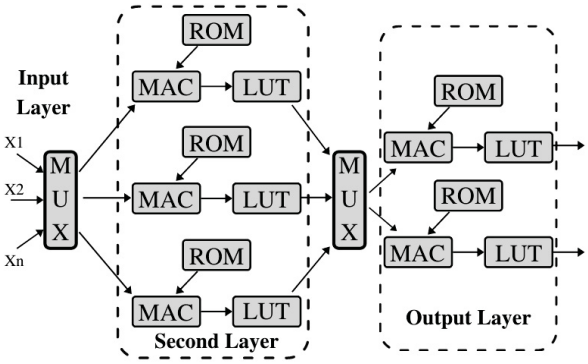
The ANN proposed by [32] was first created with the help of computer software, programmed using C language. After the NN was tuned, the weights were extracted and applied to the FPGA, specifically at the memory unit. A simple model was created to evaluate arrhythmia signals provided by an ECG scan. The model comprises one input layer, one hidden layer, and one

Figure 3.2: Units of the proposed architecture

(a) Neuron hardware model



(b) ANN architecture



Source: [7]

output layer, with 5, 3, and 2 neurons, respectively. The word length used was 8 bits long, with no specification about the fractional part. Approximately 6000 logical elements were used from these configurations, running at a $F_{max} = 16,1MHz$. Unfortunately, no meaningful discussion is presented about the results of this NN's implementation; however, this article proposes the network architecture that forms the base of further works, including the one introduced by this research.

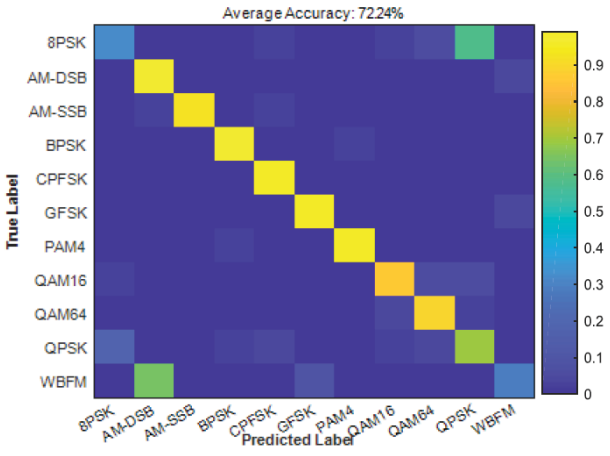
Unlike the initial architectures, the hardware NN proposed by [10] implements a 32 bits floating point network instead of a fixed point. According to the authors, this choice was made due to the floating-point numbers' greatest dynamic range. By the time it was developed, there was no embedded support to this kind of computation in the VHDL language, so the authors developed their library. When creating a floating-point hardware NN, the discussion suggested by them is based on two points: the balance between the need for a reasonable precision and that the cost increases proportionally with the higher precision. Regarding these two points, the architecture proposed by them implements a fully connected feed-forward network (MLP), where there are one multiplier and one adder per layer. This arrangement results in a combination of parallel-serially data processing computations, allied to the usage of a Read-Only Memory (ROM) where the weights are stored. This setup allows a resource economy, especially to those FPGAs that contains a relatively limited number of logical elements. The NN is based in three

layers with 2 input neurons, 3 in the hidden layer and 2 in the output (2-3-2), with a LUT-based sigmoid activation function. A simple comparison between the hardware-based NN and a software-based NN with the same characteristics is presented, where an error of approximately 6% ($\approx -24dB$) is achieved, emphasizing that the correct balance between the precision and the actual implementation can deliver results closer to the software-based implementation, with a much higher computational speed.

As the implementation of ANN in FPGAs matures, some researchers applied it in the AMC paradigm, like the work proposed by [33], where the authors combine a deep learning architecture with an FPGA Software-Defined Radio (SDR) platform. The architecture is based in a master-slave environment, where a computer equipped with a graphics processing unit works together with an SDR platform. The data used to train the network is generated with 20 samples per symbol, grouped in vectors containing 200 samples. For training, 60000 vectors are used, and 10000 for testing the network. Each of these vectors contains the raw I/Q data of the received RF network and is passed directly to the NN for classification. A combination of an unsupervised with a supervised method for training the NN is applied, with an L2 regularization in the supervised part. The raw I/Q samples can be passed directly to the NN because it consists of a CNN based on autoencoders. The CNN contains one input layer, two hidden layers, and one output layer, with no specification about the number of neurons used in each of them. The activation functions used were the ReLu (2.5) and the tanh (2.4), with a softmax (2.6) in the output layer, capable of classifying 11 modulations: 8-PSK, AM-DSB, AM-SSB, BPSK, CPFSK, GFSK, 4-PAM, 16-QAM, 64-QAM, QPSK, and WBFM, in an AWGN channel with the SNR ranging from -20 to 20 dB. The CNN was implemented through the Vivado HLS using the C++ language to generate the corresponding Verilog-HDL files, focusing on optimizing computations and memory usage/access. The resulting circuit originated an IP connected to the SDR platform using an open-source framework, the RF Network-On-Chip, that connects with the SDR and the GNU Radio, for example, for extended simulations. The CNN architecture is based on 32 bits fixed-point numbers, with no specification about the fractional part, occupying approximately 90Mb of memory. The total FPGA resource usage is around 9000 logical elements, as the FPGA used by the authors contains DSP modules integrated. The resulting architecture was tested using the AMC IP and the GNU Radio, connected with the framework mentioned above. After 640000 samples tested, the system can classify up to 70% of the samples when SNR is $\geq$0 dB, taking roughly 3μs to classify each sample. Some confusion matrix is also presented, attesting that the performance of the AMC IP implemented increases as the noise level decreases, acquiring up to an average of $\approx$72% when the SNR level is equal to 16 dB, as presented in Figure 3.3. Although this author's network model used relatively few logical elements of the FPGA, the resulting NN does not achieve a reasonable classification rate, as it needs a low level of noise in the signal for the classification (SNR $\geq$16 dB). Even so, only approximately 72% of the samples are correctly classified, probably demanding more than one prediction until the correct label is assigned to the input signal. This behavior may result from how the classification is executed, as the NN expects

to receive the raw I/Q samples of the signal. The lack of a features extraction algorithm in this architecture might contribute to the worse classification rate, as other architectures that use this approach achieve higher accuracy [7], even when the proportion of noise in the input signal is high.

Figure 3.3: Confusion matrix



Source: [33]

The promising results achieved by FPGA-based deep learning NN inspired the work in [8]. The authors implemented a feed-forward network in an FPGA based on a DeepRadio platform to develop a low-energy and low-latency hardware for modulation classification. To create the NN, a model was first created in software using 900 I/Q samples — with no information about signal impairments, from where the weights were obtained and then implemented using HDL. The model created consists of four layers: the input with 1800 neurons, two hidden layers with 100 and 20 neurons, and one output layer with 7. As the activation function, the ReLu (2.5) is used for the hidden layers, and the softmax (2.6) in the output layer is applied. The authors used the TensorFlow framework to create the network, with the Adam optimizer and cross-entropy as the loss function, as they performed a hyperparameter optimization — without specifications about the framework or the technique applied. According to the authors, to test the developed architecture, they placed a Universal Software Radio Peripheral (USRP) capable of generating the RF signal that would be classified, where they varied the transmit power to generate different SNR effects. The signal generated by the USRP was modulated into one of the following six modulations: BPSK, QPSK, CPM, GFSK, 16-QAM, and GMSK. Then, the platform containing the FPGA was placed near the USRP antennas — no information is provided about the environment where the simulation was executed — and started to sense the raw I/Q samples from the received signal from where the authors claim an average of 94% of correct classification. To achieve this result, the NN was implemented using a 16-bit floating-point, with a latency of 24μs and energy consumption of 28μJ per sample, according to the authors, resulting in 275815 logic elements and with 210 DSP modules. Despite the massive amount of neurons utilized and the lack of information about how the NN was trained, this article presents a

low-latency and low-power AMC architecture because, still according to the authors, the same NN implemented in software achieves a latency of 3,6ms and energy consumption of 36mJ per sample. Hence, it is expected to reduce more than 100 times in latency and close to 1000 times less energy consumption, attesting to the FPGA-based NN feasibility for AMC applications.

3.5 DISCUSSION ON THE STATE OF THE ART

The applications based on software implementations, not only limited to ANN and AMC, have the advantage to offer a fast development time, as it is possible to adapt the model to new requirements without much effort. On the other hand, due to the architecture implemented by most of the processors used by these applications, they lack solutions capable of implementing parallel and pipeline processing, resulting in a higher time consumed for the execution of complex tasks.

The feature-based methods applied to the AMC may require high complex calculations with a significant amount of data. These computations may require a considerable time to be fully processed, which can not be suitable for real-time applications, such as the AMC, due to the risk of losing essential information in this process. For that reason, many different strategies have been studied to decrease the complexity of the input data of AMC systems, combined with various architectures of neural networks.

One of the most promising architectures for the AMC is using a DNN as the classification method, like the one introduced by [28]. This construction can achieve a high accuracy rate, even when the noise levels achieve values near -5 dB of SNR. However, this NN is based on a large set of input features containing 21 inputs, requiring a high amount of time to be processed as they contain complex operations such as a DFT and multiple order HOS. Also, this DNN requires many neurons in its composition, summing more than 750, which would result in a high hardware consumption and computing time.

The CNN is another approach investigated to improve the AMC performance, where the raw I/Q samples from the incoming signal are used as the input to the NN. The absence of the features calculations in the classifier's input may decrease the hole model's complexity, but the accuracy rate also decays. The work of [33] highlights this behavior, as it achieves approximately 72% of accuracy in a relatively low level of noise (SNR $\geq$16 dB), which is worse when compared to other state-of-the-art architectures. The simplifications of the CNN's model may also impact the behavior in real-world scenarios, as described by [7], who tested different NN setups in more realistic channel models, with Rayleigh fading and Doppler effects, noting a significant loss of accuracy in this case.

To overcome the software implementation drawbacks, some authors have exploited the possibility of implementing NN in FPGA for the AMC task. This choice relays in this architecture's intrinsic parallel processing capabilities, combined with the fact that a specific circuit shall be constructed to optimize the tasks' execution differently from using a general-

purpose processor. The work presented by [33], a CNN which does not require the calculation of the features, is implemented into an FPGA. The architecture proposed by the author occupies approximately 9000 logical elements and can classify an incoming sample in roughly $3\mu s$. However, as discussed previously, the classification rate achieved is low compared to other state-of-the-art AMC models.

Another research that explores the FPGA resources is the one proposed by [8]. According to the authors, they can achieve an average of 94% of accuracy rate over multiple noise values. Additionally, the FPGA can classify an input sample in $24\mu s$ and consume about $28\mu J$ per sample. However, this implementation requires more than 270k logical elements of the board, as it uses more than 2000 neurons in the architecture, resulting in a circuit that is not suitable for limited-resources boards and occupying a large silicon area.

As observed from the AMC algorithms implemented so far, they use a selection of input features that do not consider each application's specificity, including both the software and the hardware solutions. Also, the FPGA AMC circuits do not explore the combination of an optimized architecture, as the NN implemented so far lacks accuracy in tiny circuits or uses a vast amount of resources to increase the results. Thus, to explore these gaps, this research proposes the investigation of the input features behavior, aiming to find the ones that can be implemented using fewer resources possible and require minimal time for the calculation. From these data, the work also proposes the optimization of the NN architecture itself, exhaustively varying its parameters to find the most optimized setup possible. This architecture will then be implemented in an FPGA to combine the software's optimization with the reconfigurable circuit's advantages.

4 AMC METHOD OF DEVELOPMENT AND IMPLEMENTATION

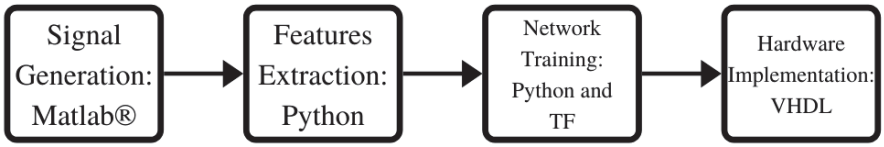
4.1 METHODOLOGY

The modulation classification system proposed is based on a synthetic dataset, as all the data used for training, validating, and evaluating both the software and hardware implementation uses the data generated by the software. The dataset is constructed using digital modulation techniques, which are corrupted by the simulation of an AWGN channel with the application of a random phase, in order to simmulte the propagation delay between the transmission and reception, which may cause a random phase between the transmitted and the received signal. Additionally, the difference between the local oscillators of the transmitter and the receiber also may lead to a difference in the signal's phase, hence the application of a random phase in the dataset aims to simulated this situations. Although the AWGN model does not fully represents a real-world scenario where wireless signals suffer from many effects, it was adopted to represent a more didatic and suitable impairment model to the research, as it enables the tests with a fixed SNR value, contrary to other impairment configurations, such as the Rayleigh fading, where the SNR varies. Also, as the AWGN model is widely used by the literature to evaluate AMC architectures [7], the combin

A few features were extracted from the data generated, and they are used as the input of the NN. These features exploit some of the signal's principal characteristics: amplitude and frequency, combined with some statistical features. Once calculated, these features are grouped by sets of each modulation and with different noise levels used to train an MLP fully-connected neural network. Since the features and the corresponding modulation are passed for the training algorithm, it characterizes a supervised learning method executed to increase the accuracy and decrease the classifications' loss. The construction of a neural network evolves many various parameters that significantly impact the overall performance. In this way, the NN's *hyperparameters*, such as the number of layers, number of neurons per layer, and activation function, were varied, resulting in a set of different configurations, from where the one with the best accuracy was chosen. After selecting the NN with the best results, its weights and biases were extracted and converted to a fixed point notation, as the hardware implementation limits the precision. These values were then passed to the FPGA, where the NN and the activation functions were implemented and evaluated with the same software-based implementation data.

The AMC's proposed architecture is based on four main steps: signal generation, features extraction, NN training and tests, and hardware implementation and tests, as observed in Figure 4.1. Each one of these steps will be discussed in the following sections.

Figure 4.1: Proposed architecture general framework

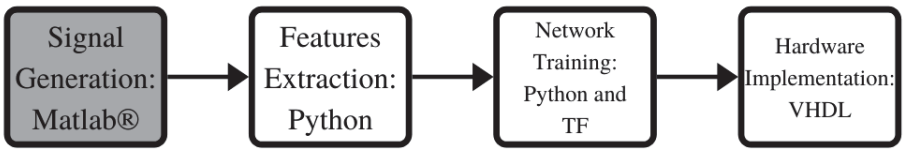


Source: The Author (2021)

4.1.1 Signal Generation

The signal generation is the first step in the architecture, as illustrated by the Figure 4.2.

Figure 4.2: Signal generation step



Source: The Author (2021)

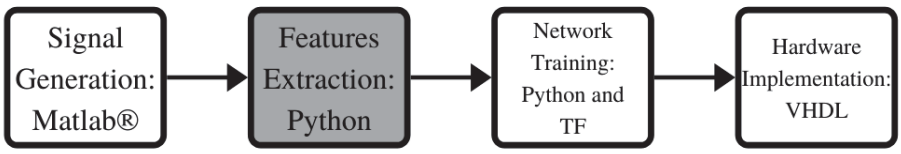
It is synthetic, based on Matlab® platform codes, due to the tool's robustness and eases to use. The generated data is created from a pseudo-random source where the following parameters are configured: SNR vector, which ranges from -10 dB to 20 dB, with steps of 2 dB; the number of frames that will be generated, in this case, 1000 were created for each modulation; the size of each frame, which consists of 2048 symbols; the symbol rate, established as 100k; the oversampling simulation factor, determined as eight samples per symbol, and the application of a random phase offset in the generated signal. These parameters are then passed to the modulator, which modulates the signal into four classes: BPSK, QPSK, 8-PSK, 16-QAM, and also generates a Noise data file. Additionally, the modulator is responsible for applying the transmission pulse-shaping filter, a raised cosine Finite Impulse Response (FIR), and applied the channel impairment on the transmitted signal, which is degraded by an AWGN model, with an SNR varying inside the previously defined range. Besides the modulation data, an AWGN dataset — the Noise data file — is created that will be used later for the NN to learn the noise characteristics. Once the signals were generated, they were converted to single-precision (32 bits) and stored in a ".mat" file for further processing with Python.

4.1.2 Features

The classification method adopted in this research is a feature-based MLP NN, which means that the incoming data should be pre-processed to extract the required information, corresponding to the "Features Extraction: Python" step illustrated in Figure 4.3.

First, the previously generated signals in the Matlab® are read using the SciPy library [34]. Then the algorithm picks each frame of each modulation and calculates the features using

Figure 4.3: Features step



Source: The Author (2021)

the Python built-in math and the NumPy [35] libraries. Once the calculation is complete, the resulting NumPy array, now containing a set grouped by modulation, SNR value, and the feature is converted to a serialized object, a Pickle file, further used by the NN Python script. The code implemented on the features extraction and the NN generation is parameterized, which means that every global option, such as the frame size or the number of frames, is centralized in a JSON settings file, making the code adaptable for changes.

The features used as the classification algorithm input were chosen after a scan was performed to reduce the number of the NN input parameters. This scan investigated how the NN would perform if some of the input data were removed, based on how good is the separation of the modulations for each possible feature from its histogram plot. From the analysis of 1000 frames, it was possible to reduce from 22 input features in the initial model to only 6:

- Standard deviation of the instantaneous frequency:

$$\sigma_{fa} = \sqrt{\frac{1}{N_c}\left(\sum_{A_n(i)>a_t} f_N^2(i)\right) - \left(\frac{1}{N_c}\sum_{A_n(i)>a_t} |f_N(i)|\right)^2}; \tag{4.1}$$

- Kurtosis of the instantaneous frequency:

$$\mu_{42}^{f} = \frac{E\left\{f_n^4(t)\right\}}{\left\{E\left\{f_n^2(t)\right\}\right\}^2}; \tag{4.2}$$

- Standard deviation of the centered-normalized instantaneous amplitude:

$$\sigma_{aa} = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} A_{cn}^2(i)\right) - \left(\frac{1}{N}\sum_{i=1}^{N} |A_{cn}(i)|\right)^2}; \tag{4.3}$$

- Cumulant order 40:

$$C_{40} = M_{40} - 3M_{20}^2; \tag{4.4}$$

- Cumulant order 42:

$$C_{42} = M_{42} - \left|M_{20}^2\right| - 2M_{21}^2; \tag{4.5}$$

- Cumulant order 63:

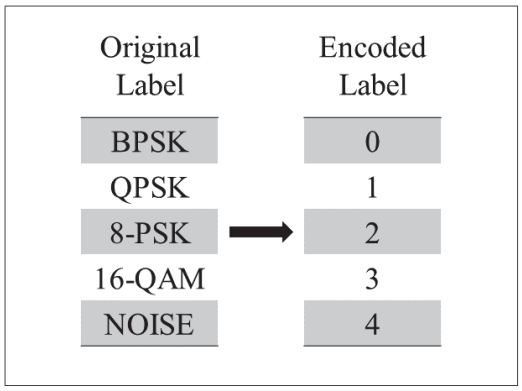
$$C_{63} = M_{63} - 9M_{21}M_{42} + 12M_{21}^3 - 3M_{20}M_43 - 3M_{22}M_{41} + 18M_{20}M_{21}M_{22}. \quad (4.6)$$

These features have a clear separation on their histogram plot when compared to the 16 remainings. The removal of the features that do not have a clear separation on its data has a significant impact on the learning process, as the NN is not capable of extracting the signal's specific characteristics from overlapped data. The six features selected histogram are shown in the Results chapter.

4.1.3 ANN's training and tests

The data passed to the NN's learning algorithm comes from the Pickle file generated by the script responsible for calculating the required features, and it is appropriately organized to improve learning efficiency. The files are read and vertically stacked, creating an array of one column containing a number of lines corresponding to the multiplication of five modulations × sixteen SNR values × number of frames. Each frame inside the features array is associated with a second array containing the respective modulation label — BPSK, QPSK, 8-PSK, 16-QAM, and Noise — that is encoded into a number — 0, 1, 2, 3, and 4, respectively, as the NN performs better with a numerical label than a textual one, as illustrated in the Figure 4.4. Finally, the data is separated into small datasets, containing the information for training and testing the NN into a proportion of 70% for training tasks and 30% for testing. Furthermore, the data is also normalized using the L2 algorithm, presented in the Scikit-Learn Library [36], as the supervised learning method adopted in this work performs better when the regularization is present [33].

Figure 4.4: NN input data label encoding



Source: The Author (2021)

The NN construction itself is made using the TensorFlow (TF)[37], a platform for machine learning developed by Google that contains a robust set of tools, libraries, and resources that enables the creation and implementation of ML models. The chosen version of TF is

implemented using Python, following the other scripts used in the research. To enable fast development and abstraction of some activities required by the TF, the NN code was implemented using the Keras [38] library, a DL API that also runs on Python. The model used for the NN creation with Keras is the Sequential, implemented using Dense layers that are fully-connected. Inside this model, the strategy adopted to overcome the over-fitting problem is to add some Dropout layers between the Dense layers, as explained in section 2.3.

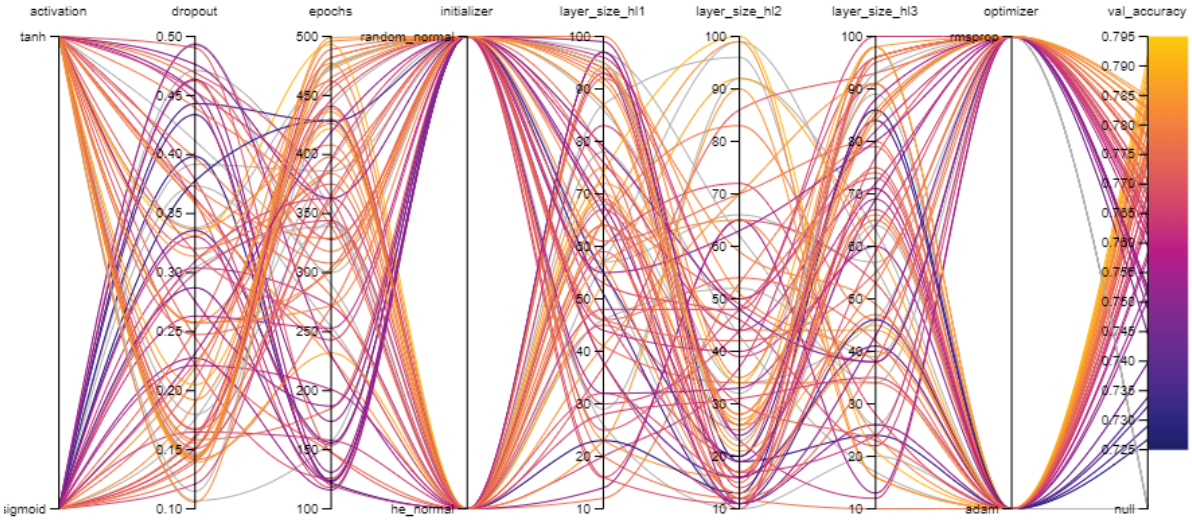
Once the model and the layer structure are defined, it is compiled, trained, and saved using the Keras functions. The NN components have some parameters that may be modified to improve the classification algorithm's performance, also known as *hyper-parameters*, that may be various activation functions, the weights and biases initialization, or the number of epochs used for training. To guarantee that the model with the highest accuracy was chosen, the hyper-parameters were extensively varied, and multiple combinations were evaluated, using the Weights&Biases(WandB) tool [39]. The WandB randomly varies the NN hyper-parameters; hence it creates a new NN architecture in each test. This model is then trained, saved, and evaluated, and its results are extracted into a web-based platform, from where it is possible to visualize and consequently select the best model of all setups tested. As multiple different models were created, each one is assigned to a unique *id* to track the settings used by WandB. The objective of the hyper-parameters tuning using the automated tool is to reduce the loss and increase the accuracy of the NN classification as the training advances in the number of epochs, executing this process many times until the best model is achieved. An example of the tuning parameters result extracted from the WandB web platform is shown in Figure 4.5.

The process of choosing the resultant architecture for the classification neural network results from the combination of two strategies: first, an architecture is chosen, and, from this initial choice, multiple models are created and evaluated until the combination of these random parameters achieved a reasonable classification. As multiple models were tested, various possible scenarios were verified, which contemplated modifications such as starting with a wide range in which the hyper-parameters may vary and progressively reduce this range, as a large number of neurons, for example, does not increase the accuracy proportionally. Another scheme also verified concerns the application of multiple activation functions, from where those with poor results were removed as the sweeps advanced, resulting in a tinier set of possible functions. In a combination of the NN hyper-parameters variation, some architectures' characteristics were also verified in this process. This verification aimed to guarantee that the chosen setup would be the most cost-effective one and contemplated changes such as a modification in the number of hidden layers, different channel impairments in the training data [1], altering the frame size, which resulted, after multiple possible configurations tested, in the restricted range applied to the hyper-parameters in combination with the architectures which showed a better performance in

[1]Both AWGN and Rayleigh fading were initially tested, but the adopted channel impairment model was the AWGN, as the Rayleigh model have caused a huge degradation in the system's performance, demonstrating that the set of data utilized was not capable, at this moment, of overcoming this intesity of signal degradation.

the initial classification results. The combination of these adjustable parameters sums up to 14 sweeps in the WandB tool, with more than 200h of training resulting in almost 1800 different NN settings, from where the best one was chosen, offering a compromise between the model's complexity and the accuracy delivered.

Figure 4.5: WandB tuning example



Source: The Author (2021)

After the NN is trained, it is validated with the validation data, and a confusion matrix is generated. This matrix contains the visible results of how the NN performed over the classification in all the tests executed, with an average correct classification score. Furthermore, the resulting NN model is also evaluated regarding its accuracy performance in the assumed range of noise levels for the input signal.

The second improvement applied to the NN was trying to degrade its setup to a point where the performance was not significantly affected, but the resource usage was reduced. The starting point was again the best architecture found by the initial set of WandB sweeps, which had the following setup:

- Droptout rate of 0.2615;
- Random-normal weights initialization, which generate tensors with a normal distribution with mean equals to zero and standard deviation of 0.05 by default;
- The activation function for the hidden layers is the tanh (2.4);
- There are three hideen layers with 50, 36, and 42 neurons each;
- The weights optimizer utilized is the Adam algorithm;
- The training loss achieved was 0.4468;

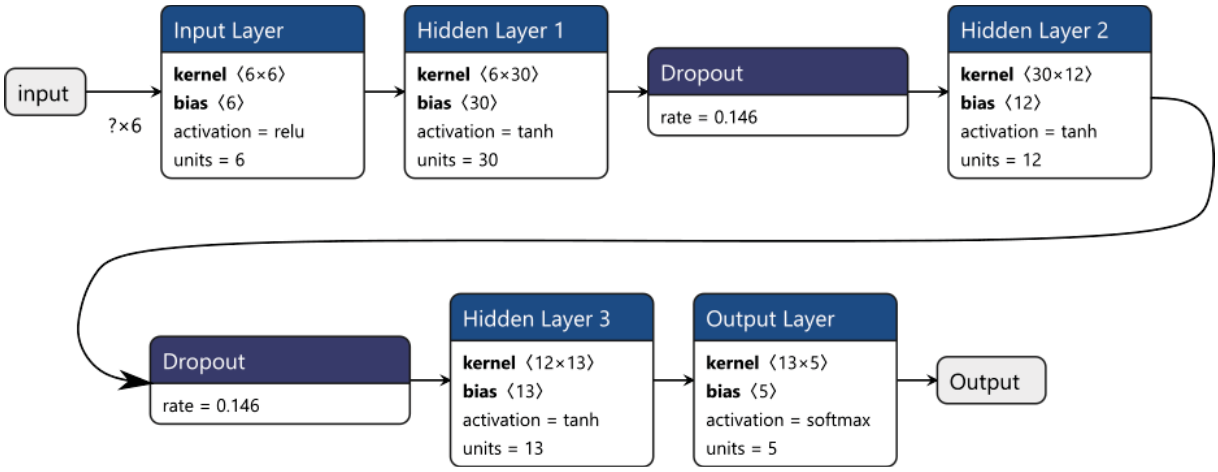
- The model's accuracy accomplished was 0.7729, an average for all the noise range.

These settings were submitted to a new sweep in the WandB tool, with a more restricted range for the number of neurons in each layer and a reduction of the possible activation functions and the learning optimizer, combined with reducing input features for the training process. This new sweep tested about 300 hundred possible combinations in just over 9 hours of testing. When completed, the resulted NN architecture had the following characteristics:

- Dropout rate of 0.1467;
- Random-normal weights initialization;
- The activation function was the tanh (2.4);
- Three hidden layers with 30, 12, and 13 neurons each;
- The weights optimizer was the rmsprop;
- Training loss of 0.3376;
- The accuracy achieved was 0.8413;

This NN architecture also used the ReLu (2.5) as the activation function in the input layer and the Softmax (2.6) in the output layer. The final NN representation is shown at Figure 4.6.

Figure 4.6: NN improved model



Source: The Author (2021)

4.1.4 FPGA realization

Unlike software-based implementation, the hardware-based NN has a series of constraints that impact the circuit's construction. The first of them is the limited resources available, as the target FPGA board has a limited number of logical elements that may be used. Second, power consumption is an essential matter, as a power-efficient circuit is desired. Third, the total area aiming to construct an IP from the generated circuit should be as tiny as possible because big circuits have a higher cost. The resulting model from the software-based should be slightly modified to meet these needs, finding a balance between the performance and the hardware requirements.

The first modification regards the precision limitation for the whole architecture. While the software-based uses Numpy floating-point numbers for calculating the features and the NN parameters, the hardware implementation used a fixed-point notation. This notation was chosen due to the saving of resources in processing these numbers, as the fixed-point notation requires fewer logical elements to process than floating-point numbers, and fewer elements are used to store these values on the board. Another modification is how the NN deals with the overflow along the calculations required for the classification, which are based on the resizing functions of the VHDL 2008 for fixed-point numbers.

After the hardware-based limitations were established, a Python script was used to extract the weights and biases for the chosen model and converting these values into a suitable fixed-point notation. From the converted values, the NN was implemented using VHDL, with the Intel® Quartus® Prime Lite as the editor for the circuit description. Once concluded, the NN was tested using the Mentor Graphics Modelsim, where the input data was the same used for the original software-based implementation, except for the conversion to a fixed point notation. As the hardware-implemented NN is theoretically the same as the software implementation, the same tests were executed, resulting in a confusion matrix and the variation of the results over different noise levels.

It is crucial to highlight that the FPGA-based NN does not calculate the features but only implements the NN itself. Hence it expects to receive the already calculated features as the input data and not the raw I/Q samples from an incoming signal, which establishes a depency of an external circuit or software to calculate the features and transmitting them to the FPGA. Because of this dependence, the total FPGA parallelism, as well as a potential time reduction of the features calculation when compared to a software implementation is not explored at this point. This limitation of the FPGA implementation occurred primarily because of the time limits established for the research. However, this step's implementation is highly stimulated for the future works of this work, as discussed in the appropriate section. Another step involved in creating the software-based NN that is not implemented in hardware is the model training process. This step applies a series of multiple optimizations using algorithms like the back-propagation and a vast amount of math operations and error adjustments, which certainly would perform

faster if implemented in a dedicated FPGA. However, due to the limited precision required for the hardware, allied to the highly complex operations applied in the training process that would require a high number of logical resources and memory, the amount of effort required to create a precise circuit, capable of training the neural network would not compensate the results achieved, which can easily be implemented in software.

The AMC NN's hardware implementation is based on the architecture shown in section 4.1.3. Assuming this is the best NN available, the objective is to adapt it to the hardware restrictions. In this manner, the hardware-based NN implements a fixed point notation for all the numerical operations, adopting a signed Q4.11 format. For convenience, manipulating the decimal numbers into the fixed-point notation used a Python library, the FXPMath.

Additionally to the limited precision, another characteristic exploited is eliminating potential hardware-cost operations, such as the square root. Although the calculation of the features was not yet implemented at this point, it is crucial to determine if such manipulations would have a significant impact on the overall performance of the NN. So for this evaluation, another sweep in the WandB tool was executed, maintaining the restrictions of the parameters applied in the previous one, but now with the fixed point notation and without the square root calculations. It is important to emphasize that although the final NN for the FPGA circuit and the software NN have some slight changes in the number of neurons, these adjustments resulted from the model's adaptation for the circuit constraints, being necessary a minor increase in its layer's neuron number. Hence, the focus was to maintain the same accuracy for both NN models, even if it contemplates testing different settings with tiny differences between them. In these conditions, the best NN architecture that should be suitable for the hardware implementation has the following characteristics:

- Dropout rate of 0.1753;
- Random-normal weights initialization;
- Tanh as the hidden layer's activation function;
- Three hidden layers with 30, 24, and 20 neurons;
- Rmsprop as the weights optimizer;
- Training loss of 0.3618;
- Accuracy of 0.8247;

The hardware-based implementation follows a top-down development, which means that the first element constructed was the neuron, then the layers, and finally the NN itself. The hardware neuron has three main components, similar to the architecture proposed by [32]: a MAC unit, RAM, and the activation function, which can be a LUT or the direct VHDL implementation.

The MAC unity is responsible for the multiplication of the neuron's input with the respective weight. Each one of the neurons inside the layers has its own MAC unity to improve the calculation efficiency. It is implemented aiming to infer the embedded multipliers of the target board for logical elements economy.

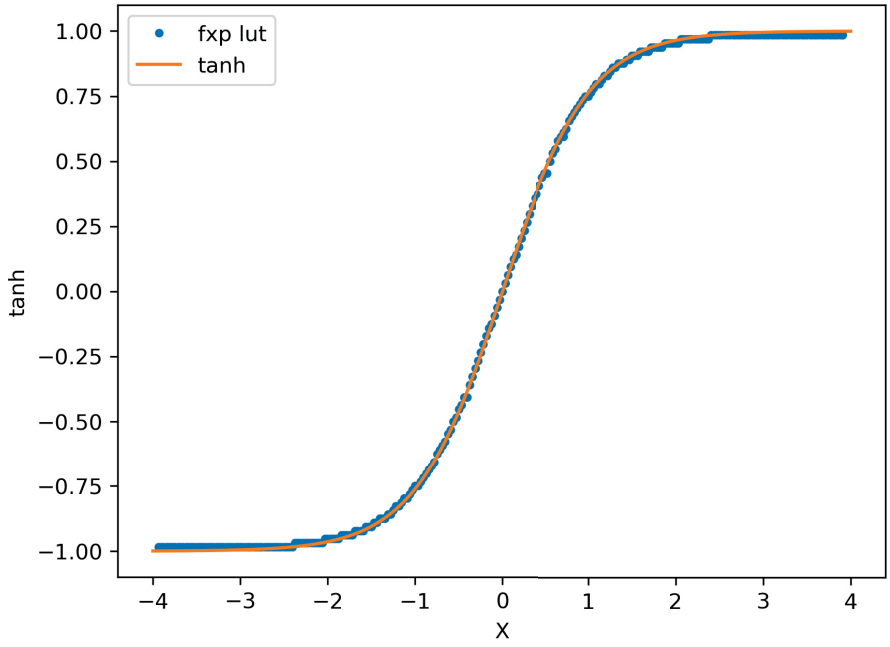
The RAM unity is where the neuron's weights are stored, so each neuron has its RAM instance. It is implemented using a VHDL array filled with the 16bits fixed-point weights, which is then inferred as a memory instance on the target board.

The neuron's activation function has three possible scenarios. The first is when the neuron belongs to the input layer, where the activation function is the ReLu. In this case, the function is directly described in the VHDL, and no LUT is necessary, as it consists only in verification if the input is bigger than zero or not. The second case regards the neuron placed in a hidden layer with a tanh as the activation function. In this situation, a LUT was implemented. The tanh LUT values were firstly generated with Numpy at Python, with 16 bits (65536 points) of data. From this set, 8 bits (256 values) were sampled. From the sampled data, it was analyzed the intervals where there was a difference between of one value and the next, utilizing as the criterion the numbers where at least one bit of difference in the fixed-point value (for a word of 16 bits, the standard value utilized in the research) was found. After these values were separated, the initial 256 values were reduced to only 95, resulting in a smaller LUT size. The LUT's response can be observed at Figure 4.7. In the VHDL, the table was implemented using an *elsif* structure, where each neuron has its own LUT. The third possible case is a neuron disposed of in the output layer. In this case, the neuron utilizes a slightly modified version of a Softmax function (2.6), that aims to avoid the exponential calculus, which would require a large amount of the board resources. This modified version verifies all the neurons of the layer and returns the biggest value, analogous to Python's MAX function. Thus this function does not provide the NN result in a probability distribution between 0 and 1, but only the result with the highest value.

The junction of these three main elements forms a basic neuron unit. The neuron's VHDL description is then responsible for instantiating the MAC, RAM, and LUT units when necessary and passing the data between them for the synaptic execution. A Moore Finite State Machine (FSM) is implemented to control the algorithm execution, as illustrated by the Figure 4.8. It is composed by the following steps:

1. *s_idle:* state where the neuron is waiting for the beginning of the calculations;
2. *s_get_weight:* the neuron request to the RAM module the respective weight;
3. *s_wait_weight:* the neuron waits for one clock cycle the response of the RAM module;
4. *s_mac:* in this state the weight and the neuron's input are passed to the MAC unit for processing;
5. *s_wait_mac:* the neuron waits for the response of the MAC unit;

Figure 4.7: LUT tanh result



Source: The Author (2021)

6. *s_mac_result:* this state verifies if the all the synapses were execute; in the positive case, it goes to the next state; otherwise, it goes back to the s_mac state and stays in this loop until all the inputs are processed;

7. *s_bias:* in this state the neuron adds its corresponding bias value to the final result of the MAC operations;

8. *s_relu:* in this state the neuron requests to the LUT the activation function value corresponding to the result of the sum of the bias plus the MAC data;

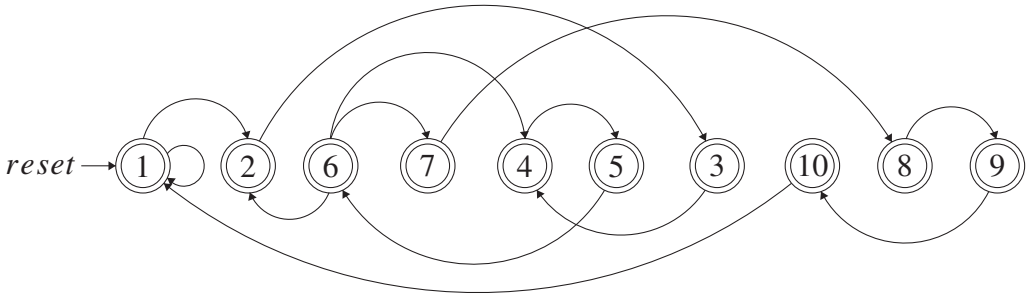
9. *s_wait_relu:* the neuron waits for the response of the LUT;

10. *s_clear:* this is the final step, where the neuron signalizes that the data processing was finished and clears the internal flags and counters, returning to the idle state.

The layer implemented in the FPGA consists of a junction of a specific number of neurons, each one with a respective memory of weights and biases. The objective is to create a container of neurons capable of applying a set of operations in the input data and transmit the result to the next layer. For doing so, the layer has one input that is connected to the input of every neuron. Hence the incoming data is processed serially but in parallel between all the neurons that form the layer. A Moore FSM is also implemented in the layer circuit to control the processing flow due to the serial input data, as shown at Figure 4.9. The layer's FSM has the following steps:

1. *s_idle:* in this state the layer is waiting for the beginning of the calculations;

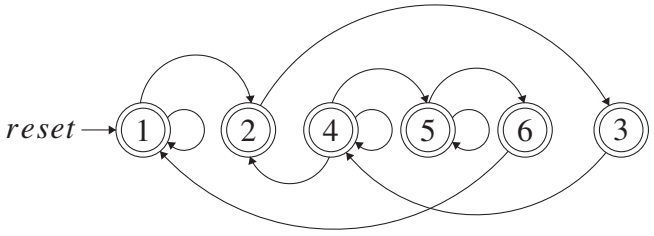
Figure 4.8: Neuron's FSM



Source: The Author (2021)

2. *s_get_data:* in this state the input of the layer is passed to each neuron;
3. *s_synapse:* this state is responsible for enabling the beginning of the calculations by the neurons units;
4. *s_wait_sinapse:* this state waits for the completion of the calculation process by the neurons and have three possible results: stay in the same state until all the neurons complete the calculus; if the number of synapses is smaller than the expected for the layer, it goes back to the s_get_data state, otherwise it goes to the next state;
5. *s_wait_activation:* this state waits for the activation function execution and hold-up until all the neurons complete the task.
6. *s_clear:* this state clears the internal execution counters and disable the neurons until a new classification is initiated.

Figure 4.9: Layer's FSM



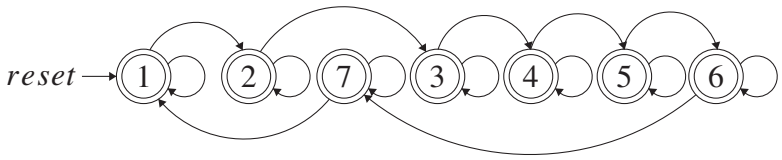
Source: The Author (2021)

Similar to the arrangement of a layer with the neurons, the top-level neural network circuit is made up of a junction of layers. Its main activity is to coordinate the data processing

flow between the layers, connecting them through a multiplexer, as the layers expect to receive the data serially. The data flow between the layers is implemented through an FSM, as illustrated at Figure 4.10. It has the following behavior:

1. *s_idle:* the NN is waiting for the enabling signal to initiate the prediction;
2. *s_input_layer:* this state enables the data processing in the input layer. It waits until the calculation is done after passing to the next stage.
3. *s_hl_1:* this state is responsible for the execution of the first hidden layer, enabling the calculation until the layer indicates that it has been finished.
4. *s_hl_2:* this state controls the operation of the second hidden layer, until it finishes the processing activity.
5. *s_hl_3:* this is the last hidden layer's state, responsible for controlling the third hidden layer execution, until it specifies that the processing has ended for in this step;
6. *s_output_layer:* this state enables the last layer of the NN, waiting for the execution to finish.
7. *s_clear:* the last state, it is responsible for clearing the internal signals and counters, returning to the idle state.

Figure 4.10: NN's FSM



Source: The Author (2021)

The junction of all those elements forms the complete neural network implemented in the FPGA. As the connection between the layers is made through the use of a multiplexer connected to all the layer's neurons, the data is inserted in the network serially and processed in parallel by all the respective layer neurons. After the calculations are completed, each result is transmitted to the next layer until the processing is executed by all the layers that compose the model. The multiplexer application instead of total parallel processing is chosen due to the logical resources economy, combined with less complex architecture. However, as not the full parallel processing potential is exploited, a slight decrease in the velocity in which the network is capable of classifying each sample is expected.

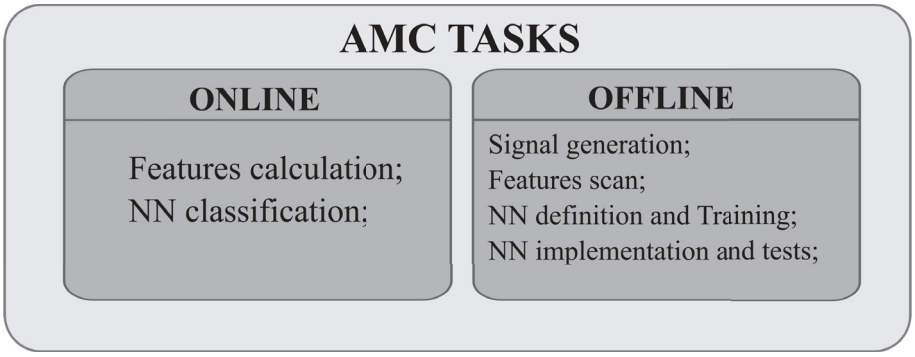
The components implemented in the NN architecture are very similar from the circuit view. The differences in the neurons, for example, are present in the weights values used and the bias of each one. Additionally, as there are many neurons with 16bit fixed-point parameter values, the manual implementation results in a significant potential for errors. In this way, a neuron model was described, and, from this model, together with a Python script, a VHDL NN file generator was developed.

The file generator is composed of a Python script that reads the previously extracted weights and biases from a NN model, already converted to the fixed-point notation. The generator then creates four VHD files categories: the neurons of each layer, instantiating the RAM unity and also the respective activation function; the RAM unities for each neuron of each layer, containing the values read from the weights file; the package file that implements the components instances; the layers file, which gathers all the neurons units. The automatically generated files were then joined in a Quartus ® project, which generates the NN's circuit.

5 RESULTS AND DISCUSSION

The resultant architecture executes a series of consecutive tasks until the final classification result is given. These tasks are grouped into two categories: the offline and the online tasks. The offline activities must be executed before the system is operating and involves the signal generation, the features scan and selection, the neural network architecture definition, training and testing, and the resultant network model's construction using the software and the FPGA. Although both implementations use the result of the offline tasks, they are executed only in software, as their implementation in hardware would require a highly complex circuit that would not achieve the same results, as discussed previously. On the other hand, the online tasks are executed after the system is constructed and when it is in operation, receiving the signal and processing it. These tasks are fully executed in the software NN, as it contemplates the features calculation and the NN classification; however, the hardware version executes only the NN classification, as the features calculation is not implemented at this point. A diagram that summarizes these tasks is presented in Figure 5.1.

Figure 5.1: AMC tasks



Source: The Author (2021)

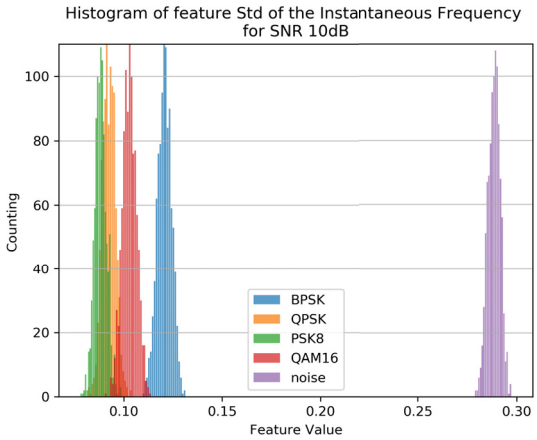
5.1 FEATURES RESULTS

The selected features applied to the AMC NN perform differently per modulation as the noise level decreases. As illustrated by the histogram presented in the Figure 5.2, they exhibit a separation performance that allows the classification of each modulation present in the signal.

The behavior that is seen in the Figure 5.2 explicit that the features have a complementary behavior in the classification task. That is, a singular feature data is not capable of classifying an incoming signal by itself; however, when combined with the other features information, the NN is capable of distinguishing each one of the possible modulations. For example, the BPSK class is not clearly distinguishable from the other modulations in the behavior presented in the Figure

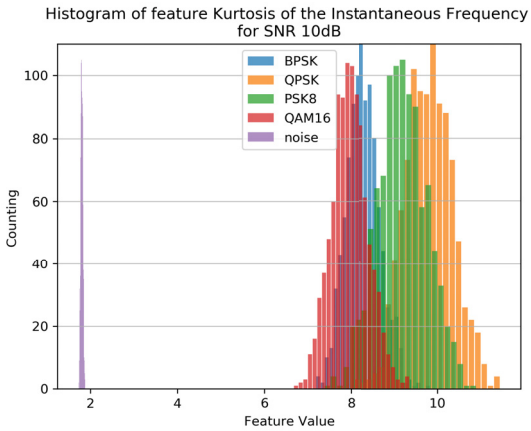
Figure 5.2: Selected features histogram

(a) Standard deviation of the instantaneous frequency histogram



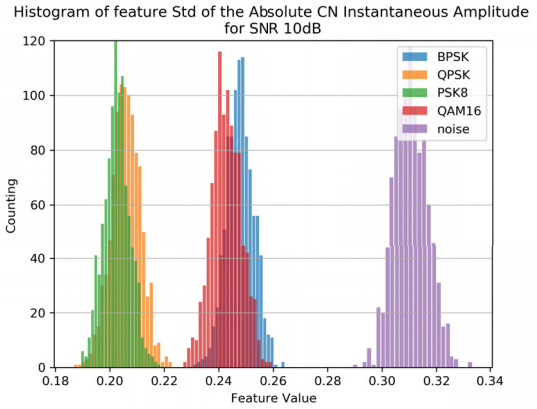
Source: The Author (2021)

(b) Kurtosis of the instantaneous frequency histogram



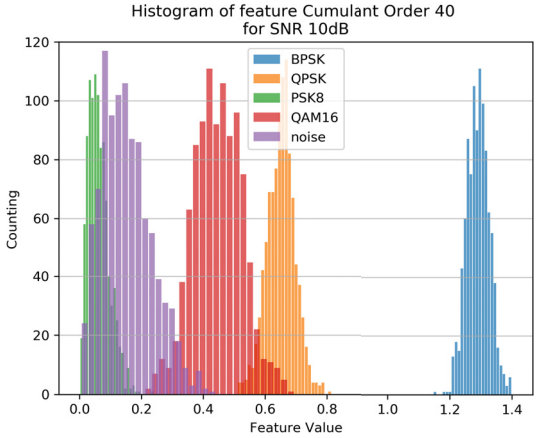
Source: The Author (2021)

(c) Standard deviation of the centered-normalized instantaneous amplitude histogram



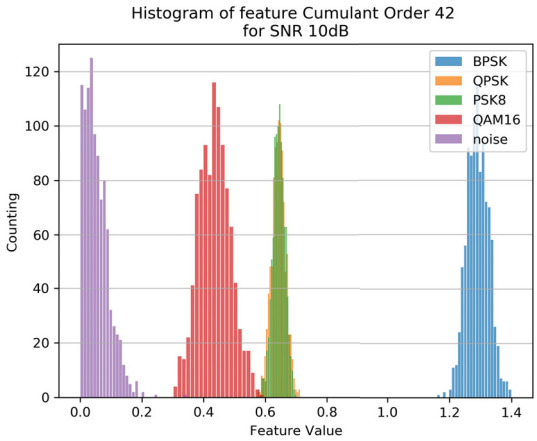
Source: The Author (2021)

(d) Cumulant order 40 histogram



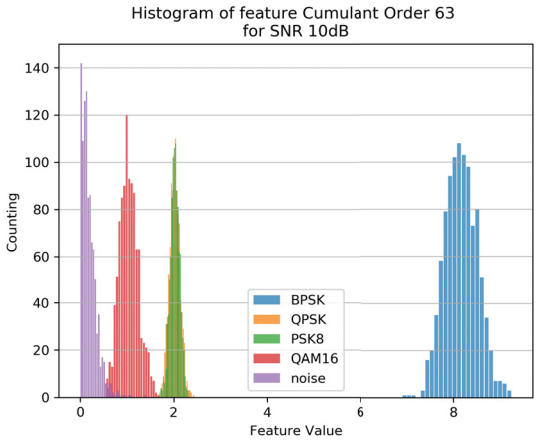
Source: The Author (2021)

(e) Cumulant order 42 histogram



Source: The Author (2021)

(f) Cumulant order 63 histogram



Source: The Author (2021)

5.2(b); nonetheless, when combined with the feature illustrated in 5.2(d), the separation from the other classes is straightforward. The features' separation nature can also be seen when the average behavior over the range of SNR is analyzed, as illustrated by the Figure 5.3.

The behavior illustrated in the Figure 5.3 indicates the average of 1000 frames over the SNR range studied, where the vertical bars represent the confidence interval, given by 3σ. This analysis corroborates with the results indicated by their histogram, as it also presents a different behavior when the noise level decreases. Additionally, the vertical lines presented indicate the three-sigma variation, which the classification task can also exploit. In a determined instant, the variation achieved by some of the features may be high enough to result in a different value from the other ones, where usually the NN could not be able to recognize the class associated with that value.

The input features set result from a scan performed in the NN's training process, from where they were reduced from 22 to the 6 ones previously presented. This set was chosen due to these features' different behavior, which may seem redundant in the first moment, as they have similar average behavior. However, when the features are combined and associated with their possible variation, the NN can identify this kind of behavior as associated with some of the specific modulations, hence improving the NN classification rate and justifying the choice of maintaining all these data, as redundant as they seem.

5.2 ANN CONFUSION MATRICES

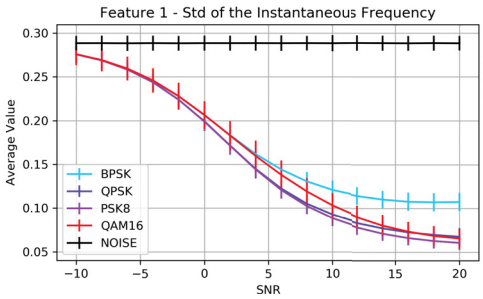
The confusion matrix provides an overview of the ANN performance, giving the average classification rate for all the tests performed. Once the training is done, the NN algorithm then executes the model's validating, with the specific dataset, resulting in the confusion matrix. The software-based NN has the result shown in the Figure 5.4(a).

From the analysis of the Figure 5.4(a), it is clear that the model achieves at least 74% of accuracy, on average, for all the possible scenarios. Additionally, it is possible to observe that the 16-QAM has the worst performance, achieving 74%, evidencing that the feature set chosen for the classification has a gap for this class considering an SNR as low as -10 dB. On the other hand, the noise is identified almost in all tests, demonstrating that the strategy of calculating the features for the noise alone and using them as input for the NN is promising, as it exhibits a different explicit behavior when compared to the modulations, as evidenced in the Figure 5.3. It is essential to highlight that the confusion matrix values average the classification, which may tend to express a lower classification rate due to the worst behavior when the NN is tested with high levels of noise in the input signal. However, as shown in the Figure 5.4(b), the NN can correctly classify approximately 90% of the input samples when the SNR is $\geq$4 dB.

The hardware-based confusion matrix is presented the Figure 5.5(a). The majority of the classification rates are lower when compared to the software implementation. These lower rates are expected due to the hardware limitations and constraints established for the implementation.

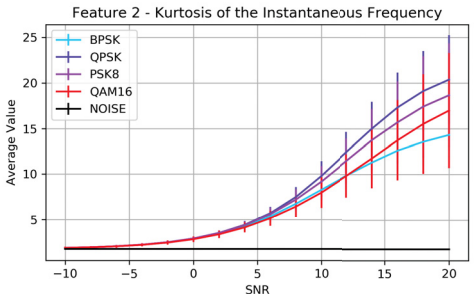
Figure 5.3: Selected features average behavior

(a) Standard deviation of the instantaneous frequency



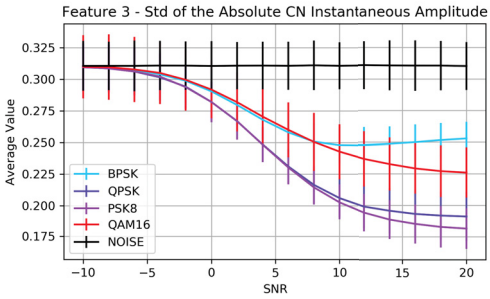
Source: The Author (2021)

(b) Kurtosis of the absolute frequency



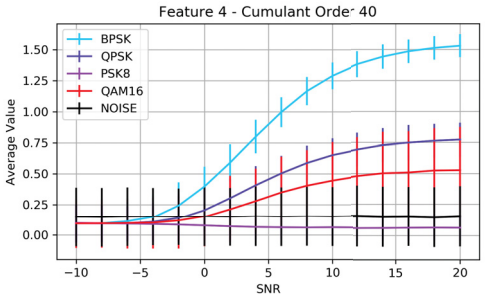
Source: The Author (2021)

(c) Standard deviation of the centered-normalized instantaneous amplitude



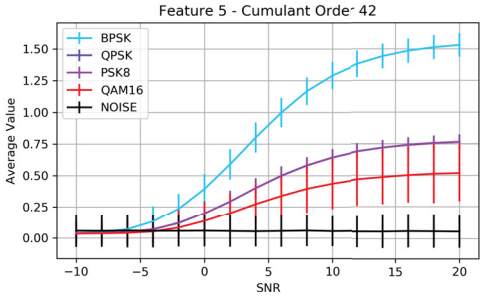
Source: The Author (2021)

(d) Cumulant order 40



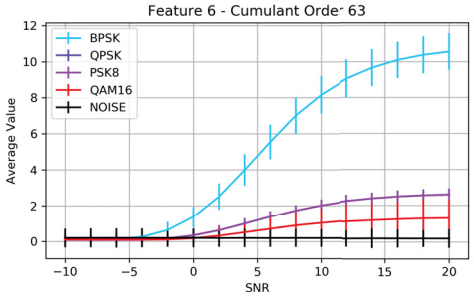
Source: The Author (2021)

(e) Cumulant order 42



Source: The Author (2021)

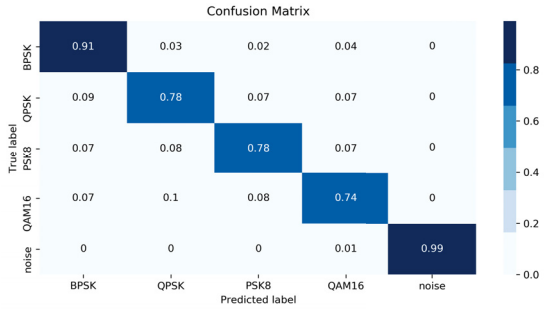
(f) Cumulant order 63



Source: The Author (2021)

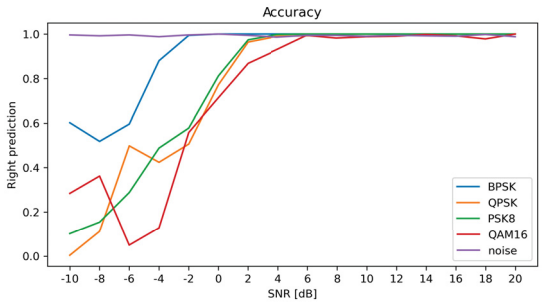
Figure 5.4: Software-based NN behavior

(a) Software-based NN confusion matrix



Source: The Author (2021)

(b) Software-based NN confusion matrix

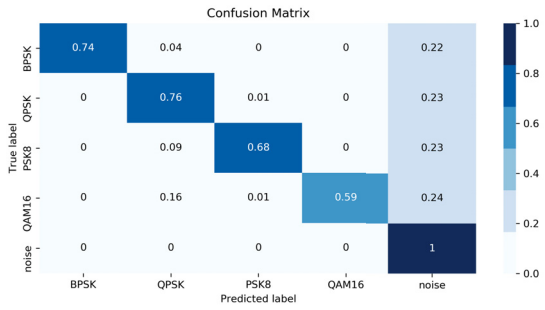


Source: The Author (2021)

Nonetheless, the same consideration is valid from the software-based implementation, as the confusion matrix exhibits an average of the NN behavior. The hardware NN's overall performance is shown in the Figure 5.5(b), from where it is clear that the classification accuracy is maintained and classified approximately 90% of the input samples when the SNR is ≥4 dB. The drawback when compared to the software implementation is the response when there is a significant noise present in the signal, specifically in negative SNR levels, where the NN quickly drops the response, mainly due to the restriction of precision of the operations and also to the error propagated along with the calculations between the network layers.

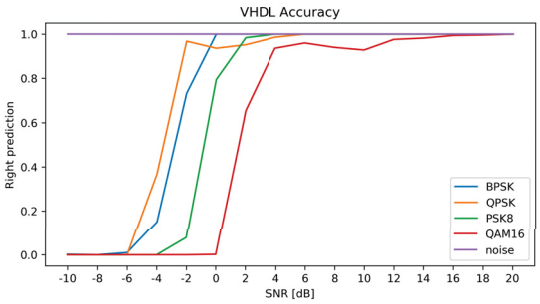
Figure 5.5: Hardware-based NN behavior

(a) Hardware-based NN confusion matrix



Source: The Author (2021)

(b) Hardware-based NN overall accuracy



Source: The Author (2021)

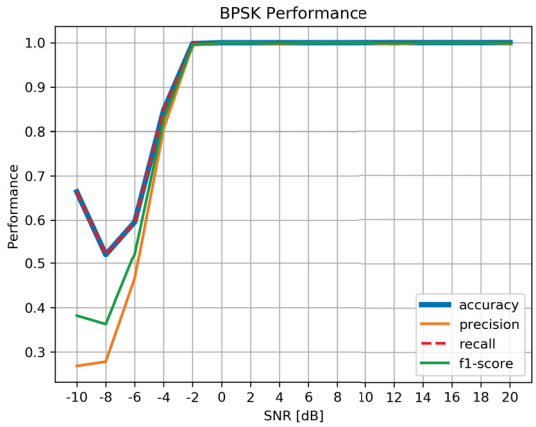
It is important to emphasize that the confusion matrix and the overall accuracy representations of the hardware-based implementation are constructed based only on the decision-making process, that is, the NN. The circuit implemented does not calculate the NN required features from the incoming signal's raw I/Q samples. Despite this limitation, the resultant hardware NN was tested using a fixed-point input data generated by a Python script, hence applying the limited precision, although the results may be harmed if the calculation process was implemented due to the imprecision errors propagation throughout the processing step.

5.3 ANN PERFORMANCE PER MODULATION

Additionally to the confusion matrix, some metrics can also be used to evaluate the performance of a neural network, being the most common the accuracy (2.7), precision (2.8), recall (2.9), and the f1-score (2.10). These metrics are presented in the next topics grouped by modulation, and a comparison between the software and the hardware-based implementation is made. The values were obtained from testing the resultant NN models with 1000 frames, over all the SNR range applied in this research. The first modulation analyzed is the BPSK, resulting in the behavior shown in Figure 5.6.

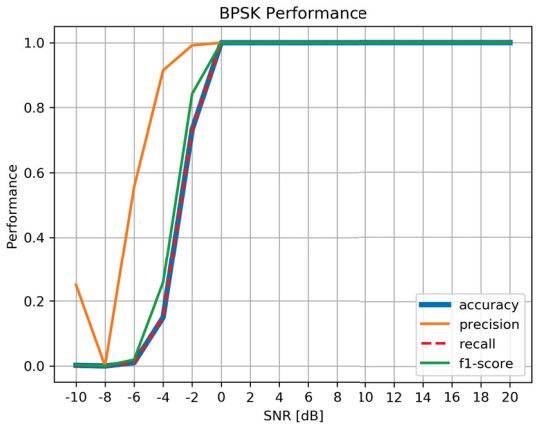
Figure 5.6: BPSK metrics behavior

(a) Software implementation



Source: The Author (2021)

(b) Hardware implementation



Source: The Author (2021)

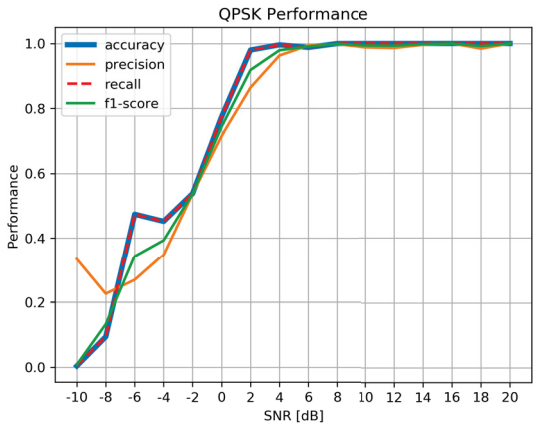
The analysis of the resultant metrics indicates that the software-based implementation has better accuracy compared to the hardware. For the software, it starts in values around 60% and rapidly increases to values near 100% when the SNR is approximately -2 dB; in the hardware implementation, the accuracy values follow the same tendency, although it achieves approximately 100% when the SNR is around 0 dB. The recall follows the same values of the accuracy for both implementations, indicating a high capacity of correct classifications. The precision also has similar behavior in both implementations, achieving almost 100% when SNR is equal to -2 dB for the software and 0 dB for the hardware-based NN. The f1-score represents a balance between precision and recall, demonstrating a better performance for the software-based NN, although the hardware-based NN also shows a similar style.

The second modulation analyzed is the QPSK, which is shown in the Figure 5.7. For this scheme, the accuracy and the recall also demonstrate similar behavior, being the software-based NN the one with a slightly better value, achieving almost 100% when the SNR is equaled to 2 dB against 4 dB for the hardware. The precision shows an increasing behavior for the hardware, which could be due to the implementation's approximations; for the software, the precision also follows the same tendency, although with a higher growth rate. As expected, the f1-score also

performs better in the software, as it demonstrates a modestly improved behavior compared to the hardware.

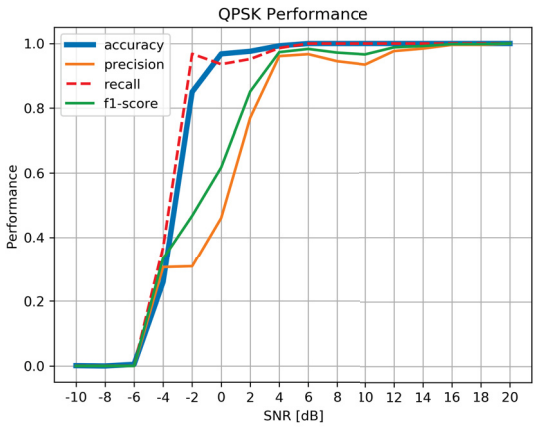
Figure 5.7: QPSK metrics behavior

(a) Software implementation



Source: The Author (2021)

(b) Hardware implementation



Source: The Author (2021)

The subsequent modulations performance is summarized in the tables 5.1 for the software version and 5.2 for the hardware version. These tables establish the minimum level of SNR required for the metrics achieve at least 90% of classification.

Table 5.1: Software Modulations Performance

Modulation	Accuracy	Precision	Recall	f1-score
8-PSK	1 dB	1 dB	1 dB	1 dB
16-QAM	3 dB	1 dB	3 dB	2 dB
Noise	-10 dB	-10 dB	-10 dB	-10 dB

Source: The Author (2021)

Table 5.2: Hardware Modulations Performance

Modulation	Accuracy	Precision	Recall	f1-score
8-PSK	0 dB	1 dB	1 dB	1 dB
16-QAM	-1 dB	3 dB	-2 dB	3 dB
Noise	2 dB	-2 dB	-10 dB	-2 dB

Source: The Author (2021)

The metrics resulting behavior demonstrate that both of the implementations have high accuracy, and the precision and recall have similar behavior, being the recall the metric with higher values for all the NN. In practice, a higher recall and a lower precision mean that the NN can correctly classify an incoming signal, although it has a higher rate of false positives due to the lower precision. For the AMC application, this is a favorable scenario, as a hypothetical secondary user would stop transmitting due to the identification of a primary user, which, in fact,

due to the false positive, would not exist. However, the scenario where higher precision and a lower recall is observed would represent an undesired situation, as the secondary user would not stop to transmit, as the classification have a higher chance to be misclassified, as the result of the higher occurrence of false negatives.

5.4 ANN OVERALL PERFORMANCE

Another analysis performed considers all the input samples of all modulations and for all the possible SNR values. All the samples were vertically stacked for this test and the evaluation metrics calculated from this big array. The tests were performed in a computer equipped with 16GB of RAM, running a Windows® 10, with an Intel® Core i7 processor and a graphic card from NVIDIA®, with 4GB dedicated. Specifically, the NN hardware version was tested using a VHDL test bench with a clock frequency defined as 50MHz, the clock of the target board (a DE2-115 from Altera equipped with a Cyclone IV FPGA). The tests performed for the software and the hardware were based on previously calculated features, which were saved in a Pickle file for each modulation and then read by the Python script or the VHDL test bench. For the VHDL, the features files were first converted to the required fixed-point notation and passed to a binary numerical system.

In this test, the software-based implementation achieves an average accuracy of 82%, requiring approximately 32ms to execute each classification (considering the output scores for each possible class). On the other hand, the hardware-based NN achieves an average accuracy of 75% and takes about 20μs or approximately 955 clock cycles to classify the same inputs. Although apparently low, the metric values represent the classification behavior of all the input samples, considering even the ones where the SNR is equals to -10dB, situation where there is a significant level of noise, meaning that the overall average rates are influenced by the good results achieved when the noise levels decrease.

If the final average accuracy of both the implementations is observed, it is clear that there is a reduction in the hardware results. This behavior may result from the constraints applied for the circuit, such as the fixed precision (Q11) adopted. Additionally, the use of lookup tables in the activation functions can result in imprecisions that accumulate as the results are being processed throughout the network, resulting in a decreased accuracy. Notwithstanding this behavior, the NN implemented in the FPGA has an outstanding classification accuracy velocity, by order of more than a thousand times faster. This velocity enables the circuit to operate in real-time applications, even though it has a diminished accuracy, which is compensated because the circuit can try to classify the incoming signal multiple times in a short period until it finds the correct label.

5.5 COMPARISON WITH THE STATE OF THE ART

Table 5.3 shows a comparison between the results achieved by this research and similar works for the software classification. The work proposed by [28] has a high accuracy; however, it

requires applying a huge NN with more than 700 neurons. On the other hand, the works proposed by [13] , [6] and [25] combine two approaches: a decision tree and an ANN. They were capable of developing models that require a reduced number of input features but demand a low level of noise in the received signal to achieve a significant level of correct classifications. The ANN model proposed by [29] also has a small NN model and uses a few more input features than the model proposed by this research. Its final accuracy is slightly better, but the input features require more complex calculations such as a DFT, for example, making it less suitable for hardware implementations. The decision tree proposed by [26] also requires a few input features but also demands at least an SNR equals 12 dB to deliver an almost error-free classification. Finally, the work proposed by [7] is the most complete one, with a large number of possible modulations and a more realistic channel impairment model. Nonetheless, it uses a prominent CNN or a big ResNet (with more than 2000 filters), both achieving only 80% of accuracy when the SNR is equaled to 10 dB.

The resultant model of this research can surpass the majority of the similar works when the number of input features is analyzed combined with the final accuracy achieved. The final model has a reduced number of neurons applied, which requires only six input features, all of them only applying statistical calculations and eliminating the necessity of a DFT, for example, which would demand more time and hardware resources to be calculated. The final accuracy achieved is ≥90% when the SNR is equaled to 4 dB, achieving an almost error-free accuracy when the SNR achieves 6 dB. However, the number of possible modulations that this model can process is reduced compared to similar works, attesting that the inclusion of new modulations in the future may increase the number of input features or even the NN architecture.

Table 5.4 presents the results of similar works in comparison to the hardware-based NN for AMC. The work proposed by [33] applies a CNN to recognize the modulation of an incoming signal. For doing so, it analyzes the raw I/Q samples in approximately 256 convolutional filters, using a fixed point notation with 32 bits in the word. The resultant NN can correctly classify at least 70% of the input samples when the SNR is equal to 0 dB and uses roughly 90000 logical resources to construct the NN circuit. The model proposed by [8] is an ANN MLP with almost 2000 neurons. This NN also expects to receive the raw I/Q samples as the input, using a fixed point of 16 bits to process the data. On average, it achieves an accuracy of 94%, with no information about the signal impairments of SNR levels informed. The circuit is constructed using over 275000 logical resources.

This research's resultant hardware model implements an ANN MLP that requires six specific features as the input to classify the incoming signal. The NN model is based on 74 neurons constructed using a 16 bits fixed-point notation and occupies approximately 37000 logical resources of the target FPGA. This model has an accuracy of ≥90% when the SNR level equals 4 dB, demonstrating that even with the reduced number of logical resources used and with more limited precision, the model is still capable of classifying a considerable portion of the input samples.

Table 5.3: Comparison with the state of the art for the software NN

WORK	MODULATIONS	CLASSIFICATION METHOD	INPUT DATA	CHANNEL IMPAIRMENT	SNR (dB)	ACCURACY (at a given SNR level)
[6]	2-PSK, 4-PSK, 2-ASK, 4ASK, 2-FSK, 4-FSK	Decision Tree	5 features	-	10;20	>90% at 10 dB; ≥95% at 20 dB
[13]	AM, DSB, VSB, LSB, USB, FM, 2-ASK, 4-ASK, 2-PSK, 4-PSK, 2-FSK, 4-FSK	ANN (4-25-7/5-10-6)	6 features	-	10;20	Analog: 95% at 10 dB; >94% at 20 dB; Digital: >93% at 10 dB; >97% at 20 dB
[25]	AM, DSB, VSB, LSB, USB, FM, 2-ASK, 4-ASK, 2-PSK, 4-PSK, 2-FSK, 4-FSK	Decision Tree and stacked ANN (7-18-18-11/1-2/1-2)	9 features	AWGN	15;20	DT: >94% at 15 dB; ANN: >96% at 15 dB
[26]	BPSK, 4-PAM, 4-QAM, 8-PSK, 4-PSK, V32, V29, V29c	Decision Tree	5 features	AWGN	-5;0;5;10; 15;20;25	97% at 12 dB
[29]	2-ASK, 4-ASK, 2-FSK, BPSK, QPSK,AM, DSB, SSB, FM, OFDM, 16-QAM, 64-QAM	ANN MLP (8-15-13)	8 features	AWGN	-5;0;5;10; 15;20;25	95% at -5 dB
[28]	BPSK, QPSK, 8-PSK, 16-QAM, 64-QAM	DNN (21-500-200-40-5)	21 features	AWGN and Rician Fading with Doppler of 100Hz and 300Hz	-5;0;5; 10;15	>99% -5 dB
[7]	OOK, 4-ASK, BPSK,QPSK, 8-PSK, 16-QAM, AM-SSB-SC, AM-DSB-SC, FM, GMSK, OQPSK, 8-ASK, 16-PSK, 32-PSK, 16-APSK, 32-APSK, 64-APSK, 128-APSK, 32-QAM, 64-QAM, 128-QAM, 256-QAM, AM-SSB-WC, AM-DSB-WC	Decision Tree, CNN (2000+ filters and 7 layers) and ResNet (2000+ filters)	I/Q	AWGN and Rayleigh Fading	-20;-15;-10; -5;0;5;10; 15;20;25;30	≥80% at 10 dB
This Work	BPSK, QPSK, 8-PSK, 16-QAM	ANN MLP (30-12-13)	6 features	AWGN	-10;-8;-6; -4;-2;0;2; 4;6;8;10; 12;14;16; 18;20	≥90% at 4 dB;

Source: The author (2021)

The IEEE 802.22 standard [40] specifies a series of requirements for the realization of spectrum sensing, especially applied to the expansion of internet services in unused television channels. According to this standard, spectrum sensing in the cognitive radio environment is used to find the blank spaces and allow the sharing of these portions of the spectrum. To operate, the standard requires a minimum level of sensitivity and SNR level, together with a probability of correct detection of 90%. These levels are defined in the Table 5.5.

From the Table 5.5 it is possible to see that the SNR range level that the resulting NN model of this research is capable of serving corresponds only to the one associated with the Analog TV, although the level required for the wireless mic sensing is close to the minimum SNR value applied in this work. However, as for the application of more advanced technologies such as the Digital TV, the system must operate at a higher level of noise, precisely when the SNR is equaled to -21 dB. For that reason, in order for the resultant software and hardware models

Table 5.4: Comparison with the state of the art for the hardware NN

WORK	MODULATIONS	CLASSIFICATION METHOD	INPUT DATA	CHANNEL IMPAIRMENT	SNR (dB)	LOGICAL RESOURCES USED	FXP PRECISION (bits)	ACCURACY (at a given SNR level)
[33]	8-PSK, AM-DSB, AM-SSB, BPSK, CPFSK, GFSK, 4-PAM,16-QAM, 64-QAM, QPSK, WBFM	CNN (1 hl) with 256 filters	I/Q	AWGN	-20;-15;-10; -5;0;5; 10;15;20	90000	32	70% at 0 dB
[8]	BPSK, QPSK, CPM, GFSK, 16-QAM, GMSK	ANN MLP (1800-100-20-7)	I/Q	-	-	275815	16	94% on average
This work	BPSK, QPSK, 8-PSK, 16-QAM	ANN MLP (30-24-20)	6 features	AWGN	-10;-8;-6; -4;0;2; 4;6;8; 10;12;14; 16;18;20	37560	16	≥90% at 4 dB

Source: The author (2021)

Table 5.5: Sensing receiver sensitivity requirents

	Analog TV	**Digital TV**	**Wireless Mics**
Sensitivity	-94 dBm	-116 dBm	-107 dBm
SNR	1 dB	-21 dB	-12 dB

Source: [40]

developed to be capable of operating in this scenario, some modifications should be performed, such as training the NN with a broader range of data in more comprehensive SNR values or the application and research of other features that should make it possible to operate in this conditions.

6 CONCLUSIONS AND FUTURE WORKS

The wireless communication field is rapidly growing in the last few years. This growth resulted in the need for new technologies capable of dealing with the scarcity of finite resources, especially regarding the use of the frequency spectrum. To address this issue, the cognitive radio and the modulation classification emerge as possible solutions, combined with machine learning, to provide a reliable and efficient use of the spectrum.

The modulation classification with the neural network model developed in this research aims to provide a cost-effective version to exploit hardware implementation characteristics. This model is made of selected features to minimize the hardware cost and the time required for the calculation, obtained through histogram behavior vs. SNR analysis. From this point, different neural network settings were tested using an automated hyper-parameter tool, resulting in the most improved possible network model. In the sequence, the model was then implemented in an FPGA, using a top-down development. Each part of the architecture — neurons, layers, and the top-level network — is a separated block with its memory and processing unity, taking into account the hardware restrictions. These singular blocks, when put together, form the hardware-based neural network.

The final AMC system developed has two configurations: the software-based NN, which utilizes Python to calculate the features and execute the classification based on a Python NN model. The second is hardware-based, which uses the Python previously calculated features data with the fixed-point precision and then have the decision-making step based on a NN implemented in the FPGA. Both of the implementations were submitted to the same tests, demonstrating an accuracy greater than 90% when the SNR is equals to $\geq$4 dB. As expected, the hardware implementation, due to the limitation of precision and the calculation adjustments, has the worst performance when the SNR is negative. On the other hand, the accuracy rapidly increases when the noise level decreases. Additionally, the hardware NN architecture can classify an incoming signal more than 1000 times faster than a similar software application, emphasizing the circuit robustness, making it suitable for a real-time application, where, although it presents a reduced accuracy when the SNR achieves negative values, it can rapdily execute a new classification attemp until the correct label is found.

The hardware implementation is capable of correctly classifying any incoming signal. However, the developed circuit depends on the input, as it cannot receive the raw I/Q samples and extract the NN required inputs by itself. The calculations needed for the features need to be adapted for the FPGA, as they will have to meet the same hardware limitations of the neural network development, like a limited precision. Besides, as the objective of the hardware implementation is to exploit the inherited FPGA's parallelism, a combination of some memory units and the calculation devices should be developed in such a way to assure the maximum

throughput and usage of resources (circuit surface or FPGA slices). For that reason, one of the system's future improvements is implementing the feature extraction directly in the circuit, which would make it possible a complete comparison with the software implementation. Another possible investigation regards the use of the multiplexers between the NN layers, that, although not directly impacts in the final accuracy, may have a tiny impact in the total time required for the NN classification. Hence the investigation of new strategies about how the connections between the layer's neurons are connected with each other is one more possible future work.

The NN implemented in this work was an MLP fully connected. This structure has a vast set of possible adjustable parameters, such as the number of layers, neurons per layer, and activation functions per layer. In the MLP NN, tiny changes can make a significant impact on the system. Hence, this study opens the possibility of testing some new architectures, like exploring a wider range of layers and neurons count or even trying other paradigms, such as a CNN. Also, some hyper-parameters were not explored in this research, like changing the batch size in the training process or altering the learning rate, which could also be combined with an even more extensive range of activation functions. Plus, the dataset utilized is made of six features selected based on their histogram and average behavior, which may result in the selection of data that is not well suited for the application. For that reason, the exploration of feature-selection algorithms that are capable of prioritizing those data that may have a better result is stimulated.

However, the resultant neural network and its hardware implementation can be applied in a cognitive radio environment where there is some signal which applies one of the modulations that the system was trained to recognize. The solution developed implements some of the most recent artificial intelligence technologies, combined with a hardware implementation capable of applying all the characteristics required for a modulation classification implemented in real-time. The system developed implements an improvement in the AMC architecture capable of surpassing some of the state-of-the-art architectures, or, at least, delivering a better cost-effective solution that requires less time to respond and uses fewer hardware resources without compromising the classification's accuracy, offering a trade-off between these characteristics.

REFERENCES

[1] J. Mitola and G.Q. Maguire. Cognitive radio: making software radios more personal. *IEEE Personal Communications*, 6(4):13–18, 1999.

[2] M. L.D. Wong and A. K. Nandi. Automatic digital modulation recognition using artificial neural network and genetic algorithm. *Signal Processing*, 84(2):351–365, 2004.

[3] O.A. Dobre, A. Abdi, Y. Bar-Ness, and W. Su. Survey of automatic modulation classification techniques: classical approaches and new trends. *IET Communications*, 1(2):137, 2007.

[4] Fang Liu, Jinkuan Wang, Yinghua Han, and Peng Han. Cognitive radio networks for smart grid communications. In *2013 9th Asian Control Conference (ASCC)*, pages 1–5. IEEE, jun 2013.

[5] Adeiza J. Onumanyi, Adnan M. Abu-Mahfouz, and Gerhard P. Hancke. Low Power Wide Area Network, Cognitive Radio and the Internet of Things: Potentials for Integration. *Sensors*, 20(23):6837, nov 2020.

[6] A.K. Nandi and E.E. Azzouz. Algorithms for automatic modulation recognition of communication signals. *IEEE Transactions on Communications*, 46(4):431–436, 04 1998.

[7] Timothy James O'Shea, Tamoghna Roy, and T. Charles Clancy. Over-the-Air Deep Learning Based Radio Signal Classification. *IEEE Journal on Selected Topics in Signal Processing*, 12(1):168–179, 2018.

[8] Sohraab Soltani, Yalin E. Sagduyu, Raqibul Hasan, Kemal Davaslioglu, Hongmei Deng, and Tugba Erpek. Real-Time and Embedded Deep Learning on FPGA for RF Signal Classification. oct 2019.

[9] J. Zhu. Towards an FPGA based reconfigurable computing environment for neural network implementations. In *9th International Conference on Artificial Neural Networks: ICANN '99*, volume 1999, pages 661–666. IEE, 1999.

[10] Suhap Sahin, Yasar Becerikli, and Suleyman Yazici. Neural Network Implementation in Hardware Using FPGAs. In *International Conference on Neural Information Processing - ICONIP 2006: Neural Information Processing*, pages 1105–1112, 2006.

[11] Simon Haykin and Michael Moher. *Sistemas de Comunicação*. Bookman, Porto Alegre, 2011.

[12] Bernard Sklar. *Digital Communications: Fundamentals and Applications*. Prentice Hall/Pearson, 2nd edition, 2001.

[13] A.K. Nandi and E.E. Azzouz. Modulation recognition using artificial neural networks. *Signal Processing*, 56(2):165–175, jan 1997.

[14] Patrick Cutno and Chi Hao Cheng. A software-defined radio based automatic modulation classifier. *Wireless Telecommunications Symposium*, 2017.

[15] S. S. Adjemov, N. V. Klenov, M. V. Tereshonok, and D. S. Chirov. Methods for the automatic recognition of digital modulation of signals in cognitive radio systems. *Moscow University Physics Bulletin*, 70(6):448–456, 2015.

[16] Xiong Zha, Hua Peng, Xin Qin, Guang Li, and Sihan Yang. A deep learning framework for signal detection and modulation classification. *Sensors (Switzerland)*, 19(18), 2019.

[17] S. Haykin. Cognitive radio: brain-empowered wireless communications. *IEEE Journal on Selected Areas in Communications*, 23(2):201–220, feb 2005.

[18] A. Hazza, M. Shoaib, S. A. Alshebeili, and A. Fahad. An overview of feature-based methods for digital modulation classification. In *2013 1st International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, pages 1–6. IEEE, feb 2013.

[19] Niranjan Muchandi and Rajashri Khanai. Cognitive radio spectrum sensing: A survey. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 3233–3237. IEEE, mar 2016.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Simon Haykin. *Neural Networks and Learning Machines*. Prentice Hall/Pearson, 3rd edition, 2008.

[22] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. `http://neuralnetworksanddeeplearning.com`.

[23] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016.

[24] Clive Maxfield. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. Newnes, 2004.

[25] E. E. Azzouz and A. K. Nandi. Automatic identification of digital modulation types. *Signal Processing*, 47(1):55–69, 1995.

[26] A. Swami and B.M. Sadler. Hierarchical digital modulation classification using cumulants. *IEEE Transactions on Communications*, 48(3):416–429, mar 2000.

[27] Muhammad Waqar Aslam, Zhechen Zhu, and Asoke Kumar Nandi. Automatic modulation classification using combination of genetic programming and KNN. *IEEE Transactions on Wireless Communications*, 11(8):2742–2750, 2012.

[28] Byeoungdo Kim, Jaekyum Kim, Hyunmin Chae, Dongweon Yoon, and Jun Won Choi. Deep neural network-based automatic modulation classification technique. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 579–582. IEEE, oct 2016.

[29] Jide Popoola and Rex Olst. A novel modulation-sensing method: Remedy for uncertainty around the practical use of cognitive radio technology. *IEEE Vehicular Technology Magazine*, 6(3):60–69, sep 2011.

[30] Zhechen Zhu and Asoke K. Nandi. *Automatic Modulation Classification: Principles, Algorithms and Applications*. Wiley, 2015.

[31] Y. Taright and M. Hubin. FPGA implementation of a multilayer perceptron neural network using VHDL. In *ICSP '98. 1998 Fourth International Conference on Signal Processing (Cat. No.98TH8344)*, volume 2, pages 1311–1314. IEEE, 1998.

[32] N. Izeboudjen, A. Farah, S. Titri, and H. Boumeridja. Digital implementation of artificial neural networks: From VHDL description to FPGA implementation. In *International Work-Conference on Artificial Neural Networks: IWANN 1999 - Engineering Applications of Bio-Inspired Artificial Neural Networks* ., pages 139–148, 1999.

[33] Zhi-Ling Tang, Si-Min Li, and Li-Juan Yu. Implementation of Deep Learning-based Automatic Modulation Classifier on FPGA SDR Platform. *Electronics*, 7(7):122, jul 2018.

[34] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[35] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[38] François Chollet et al. Keras. `https://keras.io`, 2015.

[39] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[40] Stephen J Shellhammer et al. Spectrum sensing in ieee 802.22. *IAPR Wksp. Cognitive Info. Processing*, pages 9–10, 2008.

APPENDIX A – SOURCE CODE

The source code of the AMC NN model developed is available in the Github, in the following repositories:

- https://github.com/adenilsoncastro/amcPython for the Software Model;
- https://github.com/adenilsoncastro/amc_vhdl for the Hardware Model;

The citation of this work when using the available source codes is appreciated.