HARDWARE IMPLEMENTATION FOR 802.11b/g/n SIGNAL CLASSIFICATION

by

Ramazan Çetin

B.S., Electronics and Communication Engineering, Yildiz Technical University, 2016

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2019

ACKNOWLEDGEMENTS

First of all, i would like to thank my advisor Assist. Prof. Faik Başkaya for his support and guidance. I also thank Prof. Dr. Güneş Karabulut Kurt for her precious suggestions related with communication area.

I also want to thank GOHM Electronics company and my colleagues Cem Ayyıldız and Mustafa Said Uçar for their assistance. Finally, i am very grateful to my family for their continuous patience and support.

ABSTRACT

HARDWARE IMPLEMENTATION FOR 802.11b/g/n SIGNAL CLASSIFICATION

The number of devices which can communicate wirelessly are increasing with the technological improvements day by day. This situation causes congestion on electromagnetic RF spectrum. RF spectrum should be utilized efficiently due to the fact that it is highly limited. Cognitive Radio (CR) detects primary user signals in spectrum and lets secondary users transmit their signals when spectrum is free. CR achieves detection of primary user signals by spectrum sensing techniques such as energy detection, cyclostationary, matching filter, radio identification and classification based methods. Radio identification provides more information than the others by detecting transmission technology of signal. In last years, machine learning and deep learning based methods are also used in spectrum sensing tasks. Success possibility of deep learning extremely depends on data. Therefore, deep learning shows strong performance on radio identification applications due to obtaining huge amount of data from Software Defined Radios (SDR). In this work, Convolutional Neural Network (CNN) based radio signal classifier hardware is designed using Universal Software Radio Peripheral (USRP) E310 embedded SDR. Most of the works in literature use SDRs as slave devices that run connected to computer. However, embedded hardware is small sized and brings opportunity of field deployments by running standalone. Designed hardware can classify 802.11b/g/n signals blindly. Classification process starts with scanning spectrum and obtaining spectrogram image which is time-frequency transformation of spectrum. Generated spectrograms are fed into the CNN based classifier and prediction of input is achieved. CNN based classifier is trained previously by collecting 802.11b/g/n signals from wireless modem. Our work can classify LTE, GSM or other signals due to adaptivity of deep learning by training CNN with those signal samples.

ÖZET

802.11b/g/n SİNYAL SINIFLANDIRMASI İÇİN DONANIM TASARIMI

Kablosuz olarak haberleşebilen cihaz sayısı, gelişen teknolojiyle birlikte gün geçtikçe artmaktadır. Bu durum elektromanyetik RF spektrumda sıkışıklığa yol açar. RF spektrum kısıtlı olduğundan dolayı verimli bir şekilde kullanılmalıdır. Bilişsel radyo (BR), birincil kullanıcıları tespit eder ve ikincil kullanıcıların spektrum boş olduğu anda sinyallerini göndermelerine olanak sağlar. BR, enerji algılama, çevrimsel durağan özellik algılama, uyumlu süzgeç, radyo tanımlama ve algılama gibi metodları kullanarak birincil kullanıcıların tespitini sağlar. Radyo tanımlama, gönderilen sinyalin teknolojisini tespit ettiğinden dolayı diğer metodlara göre daha fazla bilgi sağlar. Son yıllarda, makina öğrenmesi ve derin öğrenme metodları spektrum sezme uygulamalarında kullanılmaktadır. Derin öğrenme Yazılım Tabanlı Radyo(YTR) ile sağlanan çok büyük miktarda veri sayesinde, radyo tanımlama uygulamalarında güçlü performans sergilemektedir. Bu çalışmada, USRP E310 gömülü YTR'si kullanılarak CNN tabanlı sinyal sınıflandırıcı bir donanım tasarlanmıştır. Literatürdeki birçok çalışmada YTR'ler bilgisayara bağlı köle cihazlar olarak kullanılmaktadır. Buna karşılık, gömülü donanım boyut olarak oldukça küçüktür ve bağımsız olarak çalışarak saha dağıtımına olanak sağlar. Tasarlanan donanım 802.11b/g/n sinyallerini sınıflandırır. Sınıflandırma işlemi spektrum tarama ve spektrumun zaman-frekans dönüşümü olan spektrogram resimlerini elde etme ile başlar. Bazı işlem adımlarından sonra, oluşturulan spektrogram resimleri CNN tabanlı sınıflandırıcıya giriş olarak verilir ve girişin tahmini yapılır. Derin öğrenme tekniklerini kullanmak, sistemin adaptasyonunu güçlendirir. Örneğin, CNN tabanlı sınıflandırıcı uygun sinyallerle eğitilerek, çalışmanın LTE, GSM veya diğer sinyalleri sınıflandırması kolayca sağlanabilir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

b_i	Bias of Neuron
f_s	Sampling Frequency
z_i	Output of Neuron
w_i	Weight of Neuron Connection
μJ	micro joule
∂	Partial Derivative

LIST OF ACRONYMS/ABBREVIATIONS

Adam	Adaptive Moment Estimation
AdaGrad	Adaptive Gradient Algorithm
ADC	Analog-to-Digital Converter
AGC	Automatic Gain Control
AMC	Automatic Modulation Classifier
ANN	Artificial Neural Network
API	Application Programming Interface
ARM	Advanced RISC (Reduced Instruction Set Computer) Machine
ASK	Amplitude-Shify Keying
ASM	Active Shape Model
AWGN	Additive White Gaussian Noise
AXI	Advanced Extensible Interface
CE	Computation Engine
CHDR	Compressed Header
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CR	Cognitive Radio
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
DAE	Data Acquisition Engine
DC	Direct Current
DDC	Digital Down Converter
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
EMG	Electromyogram
ENOB	Effective Number of Bits
FC	Fully Connected
FFT	Fast Fourier Transform
FIR	Finite Impulse Response

FPGA	Field-Programmable Gate Array
GMSK	Gaussian Minimum Shift Keying
GPP	General Purpose Processor
GPU	Graphics Processing Unit
GRC	GNU Radio Companion
GSM	Global System for Mobile
IC	Integrated Circuit
ICI	Inter-Carrier-Interference
IF	Intermediate Frequency
IIR	Infinite Impulse Response
IP	Intellectual Property
IQ	In-Phase/Quadrature
ISM	Industrial, Scientific and Medical
IoT	Internet of Things
LNA	Low Noise Amplifier
LO	Local Oscillator
LTE	Long Term Evaluation
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
MAC	Media Access Control
MATLAB	Matrix Laboratory
MCS	Modulation and Coding Scheme
MIMO	Multiple-Input Multiple-Output
ODMA	Opportunity Driven Multiple Access
OFDM	Orthogonal Frequency Division Multiplexing
OS	Operating System
PCI	Peripheral Component Interconnect
PC	Personal Computer
PHY	Physical Layer of OSI Model
PN	Pseudo Noise
PSK	Phase-Shift Keying
QAM	Quadrature Amplitude Modulation

ReLU	Rectified Linear Unit
RFNoC	Radio Frequency Network on Chip
RF	Radio Frequency
SDR	Software Defined Radio
SNR	Signal-to-noise ratio
SS	Spectrum Sensing
STFT	Short Time Fourier Transform
SVM	Support Vector Machine
SoC	System on Chip
UDP	User Datagram Protocol
UHD	Universal Hardware Driver
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VHDL	VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
VHF	Very High Frequency
WLAN	Wireless Local Area Network

1. INTRODUCTION

Although electromagnetic RF spectrum is a limited and valuable resource for all users, it is overly congested due to increasing number of electronic devices such as mobile phones, wireless sensors, smart watches, etc. [7]. Besides that, available spectrum is not utilized efficiently [8]. Technological improvements have introduced CR, that provides efficient usage of available spectrum using Spectrum Sensing (SS) techniques. Two important words in CR are primary and secondary users. Primary and secondary are the users who have a higher priority to use specific frequency bands and the users who have lower priority over primary users, respectively [9]. CR tries sensing multidimentional regions in frequency, time and space [10] that is not utilized by primary users which is called spectrum holes and provides spectrum access using these gaps. CR solves network crowding problems by identification and classification of primary user signals in available spectrum [11] and lets secondary users employ idle frequency bands. Emergency communications [12], dynamic spectrum access [13], military and public security, home and indoor communications and vehicular communications [14] are some of the application areas of CRs.

SS, which acquires awareness about frequency spectrum usage and detects primary user signals, is one of the significant features of the CR [15]. On the other hand, it is better to collect more information about primary users, thus it is important to detect signals but classification of signals is also another significant task [16]. In general, SS consists of detecting spectrum holes, predicting spatial directions of interferers, finding spectral resolution of spectrum holes and achieving signal classification [7]. In this study, an embedded signal classification hardware for cognitive radio applications was designed.

SS is generally achieved by five methods, which can be listed as energy detector based, waveform-based, cyclostationarity based, radio identification based and matching filter based sensing [9]. In energy detection based sensing, signal is detected and compared with a pre-defined threshold to find active primary users. Determining the

threshold and bad performance in low Signal-to-Noise Ratio (SNR) are main challenges of this method [9,17]. Waveform-based sensing detects known patterns such as preambles, pilot signals, spreading sequences etc. in received signal. This technique is only valid for known signal patterns. In cyclostationarity based sensing, a periodic patterns in signal that is not available in noise are searched. This system can differentiate between noise and primary user signal. Radio identification based sensing provides total information about characteristics of signal by identifying transmission technology. This type of information lets cognitive radio perform better due to extended knowledge about primary user signal. In radio identification based sensing, distinguishable features of target signals are extracted and used for classification of signals. In [18], neural network based identification is performed using channel bandwidth and shape of signal as features. Finally, in matching filter based sensing, received signal is correlated with a known pattern of primary user signal to find a match. This method needs complete knowledge about primary users' signals [9].

CR devices require configurability in radio hardware in order to change its physical layer properties according to target primary user signals. Configurability in radio hardware is not a new term; in 1980s, some configurable receivers were designed [19]. At present, configurable radios are called as software defined radio. A SDR is a hardware whose physical layer features such as frequency, gain etc. can be configured by software [20]. SDR can change its modulation, frequency, analog-digital filter frequencies in run-time. SDR receives analog signals with configurable center frequency and bandwidth, and digitize them using analog-to-digital converters (ADC). After passing signal to digital domain, any signal processing technique such as demodulation, filtering, interpolation, decimation etc. can be performed in SDR using software. In this context, CR is also an SDR device, which senses spectrum, and shows reaction according to data that is collected from the air.

Thanks to huge available data, lots of algorithms are developed for learning by unsupervised and supervised methods, recently. These methods learn features from data such as images and videos and predict result of unseen data for testing how good the algorithm learned features from training dataset [21]. Deep neural networks as a

learning method performs well on several machine learning tasks such as visual recognition, audio recognition, natural language processing etc. Deep neural networks are also employed by researchers in cognitive radio applications such as [22–25]. Employing deep learning in communication system problems such as radio signal identification has several advantages. Deep learning requires huge amount of data, which can be easily acquired from radio systems. Deep learning performs automatic feature selection by learning the system from data, which will avoid us from a challenging manual feature selection task [24].

In this work, an embedded real-time signal classification hardware called as BGN-C for 802.11b/g/n WLAN signals is designed. USRP E310 is used as an implementation hardware. USRP is an open source SDR hardware from Ettus Research [26]. Software of SDR is designed using other open source projects GNU Radio, Universal Hardware Driver (UHD) and RF Network on Chip (RFNoC). BGN-C employs radio identification based SS on 802.11b/g/n signals. Device collects samples from spectrum and transforms samples to spectrograms which are time-frequency representation of spectrum. These spectrograms are classified by designed CNN. Finally, generated CNN model is implemented on hardware, which has limitations, for real-time classification of signals.

The contribution of this work is to design a signal classification task in an embedded standalone device that is suitable for field applications thanks to its small size. Our intention is to improve the prediction time of the device and use it in real applications.

1.1. Related Work

SS lets radio identify frequencies that are available for transmission without the interference of other signals [27]. This method is used in CRs to detect primary user signals in the RF spectrum and allows secondary users to employ idle frequencies for their transmissions. One technique that is used in spectrum sensing is the energy detection method, which is employed in [28–31], but this method is much susceptible to SNR. Also, authors used other methods for spectrum sensing such as matched filter based [32–34] and cyclostationarity based [35–37]. Matched filter based methods require full knowledge about user signals and also demodulation of signals. The demodulation process brings lots of challenges like time and frequency synchronization of carriers [29]. Also, another method of spectrum sensing is radio signal identification, which provides more information about user signal [9]. For example, if cognitive radio detects primary user signal is Bluetooth; it can extract more information like the signal range.

In our work, radio signal identification based spectrum sensing is used to differentiate 802.11b/g/n WLAN signals. 802.11b/g/n signals are chosen as target signals for our system. The proposed system can also be extended to other signals such as LTE, Bluetooth, GSM, some IoT protocol signals, etc. This system is targeted to be employed in CR applications. Radio signal identification and classification problem is not new in the communication area and it is already studied by many researchers. In [38], fundamental parameters of OFDM transmission is blindly estimated using an algorithm that extracts cyclostationarity characteristics of signals. It is shown that the designed detection algorithm can estimate under AWGN and multipath fading channels with limited data. Rebeiz *et al.* designed an energy efficient processor for modulation classification using second-order cyclostationarity features of the detected signal in [39]. The processor first detects if the signal is single or multi-carrier, then it classifies the signal as M-QAM, M-PSK, M-ASK, GMSK, OFDM or DSSS. The processor achieves 95% accuracy at 10 dB SNR with $10.37\,\mu J$ total power consumption. In [40], an automatic modulation recognition algorithm is designed using higher-order cyclic cumulants of the signal. Also in [41, 42], authors employed fourth order cumulants of signal for

modulation classification.

Apart from the traditional signal processing methods, machine learning based techniques are also employed in signal classification problems in recent years. Machine learning based SS has some advantages over traditional methods. The training phase of the network is autonomous and it does not require any prior information about signals. Also, machine learning techniques can learn the patterns implicitly; this is more adaptive to new data compared to classic methods [43]. Park *et al.* proposed a Support Vector Machine (SVM) classifier, which uses the wavelet transform information of signals to classify 8 types of modulations [44]. The classifier achieved 95% accuracy on a wide range of SNR. Also in [45,46], SVM based classifiers are used for modulation classification tasks. Zhang *et al.* employed random forest based classifier, which is fed by five kinds of entropy features for automatic modulation recognition task [47]. In [48], the K-means clustering based method is used for detecting white spaces in the spectrum. The algorithm is implemented on real hardware using MATLAB and Xilinx System Generator. Genetic algorithm with K-Nearest Neighbor is used in [49] for Automatic Modulation Classifier (AMC), which classifies BPSK, QPSK, QAM16, and QAM64 modulations. Higher order cumulants are used as features for algorithms.

Most recently, deep learning also attracts authors for communication tasks such as automatic modulation classification, radio signal classification, and SS. There have been many publications on radio signal classification using Artificial Neural Network (ANN) and CNN are released, recently. Deep learning techniques are improved due to the availability of the huge amount of data and the advanced hardwares, which can perform parallel computation such as Graphics Processing Unit (GPU) and Field-Programmable Gate Array (FPGA). Deep learning is superior over machine learning in feature extraction since it can acquire features from inputs itself and optimizes its parameters [50]. In our work, we used a convolutional deep neural network for feature extraction and classification of 802.11b/g/n signals. In [51], Jagannath *et al.* designed ANN based classifier for an automatic modulation classification task. Filters, automatic gain control (AGC) and frequency offset correction are employed as pre-processing steps. Amplitude, phase, frequency, moments and cumulants are used

as features for ANN. This classifier is implemented on the PC and tested using USRP devices to classify seven different modulation types with various transmit powers. Wu *et al.* proposed five layers of CNN for the Very High Frequency (VHF) radio signal classification problem [52]. The accuracy of the network is about 95% on data, which is simulated on MATLAB with low SNR. However, the model is not successful in real data. The accuracy for real data is 55% in data mode and 76% in speech mode due to environmental effects such as frequency offset, phase noise, etc. In [53], authors achieved modulation classification using two well-known CNN structures as GoogleLeNet and AlexNet. They used different representations of radio data such as constellation diagram, gray image, enhanced gray image and 3-channel image as input to CNN. They compared CNN and other methods, such as cumulant and machine learning based on the same dataset. As a result, CNN has better classification accuracy and advantage of automatic feature extraction. Riyaz *et al.* proposed a CNN based RF fingerprinting method using I/Q sequences as input [54]. It identifies different devices using unique physical layer signatures, which are transmitter-receiver pair characteristics and channel effects. Their method outperformed machine learning based methods such as SVM and logistic regression. In [22], authors designed a ResNet based modulation classifier, which differentiates 24 types of modulation schemes and compared it with their previous designed CNN. ResNet showed better performance on the same conditions. They used USRP B210 to transmit and receive signals for their dataset. Also, the authors employed CNN based modulation and signal classifier in [50, 55, 56].

Deep learning has many superiorities compared to other traditional and machine learning based techniques. A most important aspect of deep learning is its training dataset. Thus, potential attacks on classification tasks cause a significant decrease in the accuracy of network [57]. Recently, some authors have researched about effects of jamming attacks on deep learning tasks. In [57], it is stated that deep learning based algorithms are exceedingly sensitive to jamming attacks, and this introduces some security concerns. They used white and black box attacks on a deep learning based classifier and resulting in a dramatic reduction in accuracy. On the other hand, in [58], authors developed defense methods for attacks on deep learning based classifiers. They used Peak-to-Average-Power-Ratio values and the probabilities in the last layer of the

network to catch dispersion shift generated by the attacker. Also, in [59], a security method is developed for CNN based AMC which suffers from attacks on its inputs.

1.2. Outline and Motivation of the Thesis

In this thesis, the main focus is designing a low cost standalone real-time signal classifier, which will help to sense the spectrum in CR applications. 802.11b/g/n WLAN signals are chosen as target signals for our classifier but the system can easily be extended to other signals such as LTE, GSM, etc. Most of the works in this area are employed by USRP hardware, which is connected to the PC. This is not suitable for practical applications. In our work, USRP E310 is used as classification hardware, which is extremely small and has its own CPU but also has lots of limitations. We designed a CNN based classifier on E310 and tried to overcome hardware restrictions.

This thesis is organized as follows. Background related to SDR and target signals of designed system 802.11b/g/n is summarized in Chapter 2. In Chapter 3, hardware and software tools that are used in the design of the system are explained in detail. In Chapter 4, the entire device architecture that consists of Data Acquisition Engine (DAE), pre-processing steps and CNN architecture is given. Real-time tests of device and classification results are presented in Chapter 5. In Chapter 6, conclusion and future work are given.

2. BACKGROUND

In this chapter, the background of the proposed work will be given to help understanding better the following sections. In our work, IEEE 802.11 b/g/n signals are classified in real-time using an embedded SDR hardware. Thus, firstly definition of SDR, its architecture and operation will be explained first. In the next section, physical layer features and differences between 802.11b/g/n signal standards will be given.

2.1. Software Defined Radio

Wireless communication technologies such as voice, data, video broadcasting and secure communications wave become more and more important with the rapid improvement in electronic technology. At this point, a technology called Software Defined Radio emerged to accelerate and facilitate wireless communication issues by its configurable hardware.

In the past, most of the radios have designed to operate for a specific signal type, and parameters of radio such as modulation type, coding, sampling rate are defined before design [20]. Improvements in technology drive the demand for a radio system that supports different signal types. In Wireless Innovation Forum, SDR was accurately defined as "A software-defined radio is a radio in which some or all of the physical layer functions are software defined" [60]. A basic idea of SDR technology is that radio components like modulators, mixers, demodulators, filters, etc. can be fully configured and designed by software or embedded system [20]. To sum up, engineers can focus more on designing complex communication systems instead of the hardware by utilizing the configurability of SDRs. Today, SDR is employed in very broad applications in commercial and academic domains such as Dynamic Spectrum Positioning, Opportunity Driven Multiple Access (ODMA), Cognitive Radios, Security Critical Communications, Cooperative Wireless Network Diversity [61]. Basic SDR architecture is shown in Figure 2.1.

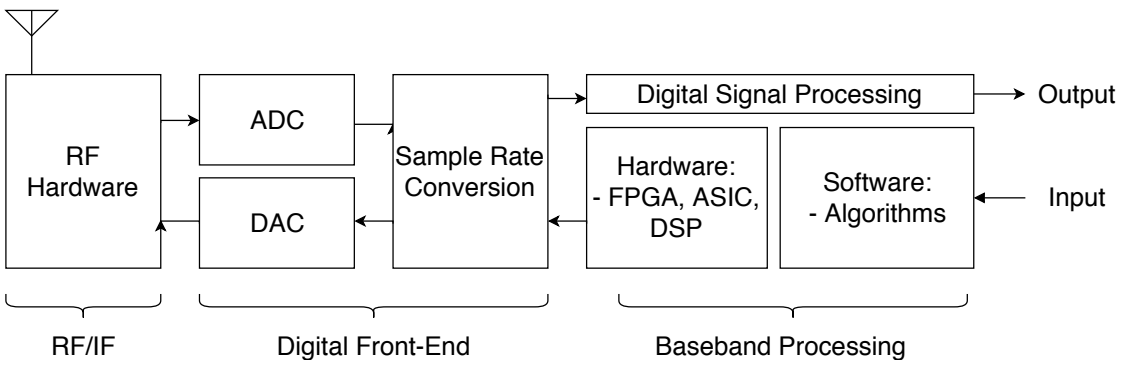
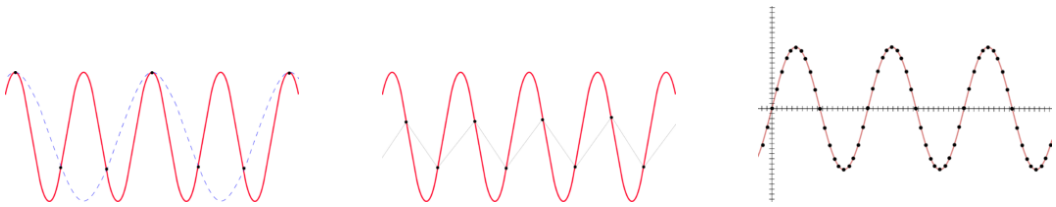


Figure 2.1. Basic SDR Architecture.



(a) Sampled by $1.5f_s$. (b) Sampled by $2f_s$. (c) Sampled by $20f_s$.

Figure 2.2. Signals with different sampling rates [1].

SDR includes analog front-end which consists of analog filters, mixers, and analog amplifiers. The received high-frequency RF signal is mixed and converted to Intermediate frequency (IF) which is an inter-stage between RF and baseband. Its frequency is generally a few hundred kilohertz. When the RF signal is converted to IF, it can be digitally converted to baseband using ADCs. If RF signal is directly converted to baseband in front-end, the signal can be corrupted because of imperfections and DC offsets of Local Oscillator (LO) and mixer. Therefore, changing the frequency of RF signal to IF first is better than converting to baseband directly.

ADCs play a significant role in SDR. ADC performance is measured with several parameters: SNR, Effective Number of Bits (ENOB), spurious-free dynamic range (SFDR) and power dissipation [62]. The bandwidth of the received signal depends on the sampling rate of ADC. According to Nyquist theorem real signals should be sampled at least twice of their frequency to represent them in the digital domain without corruption due to aliasing. When the signal is complex, this rule can be converted to this: analog data should be sampled at the rate of frequency of the signal by ADC. But in practical applications, the signal should be sampled more than two times of frequency to represent its amplitude better. In Figure 2.2 three examples of analog signal sampling were depicted. In Figure 2.2a signal is sampled by $1.5f_s$, which causes aliasing on the obtained signal. So, the frequency of acquired signal looks lower than twice of its real frequency due to the aliasing effect. In Figure 2.2b signal is sampled in the Nyquist rate which is $2f_s$ and signal is created without aliasing. But it still does not represent the true form of signal. The obtained signal's amplitude is corrupted, but the frequency is the same as the original signal. In Figure 2.2c, the

analog signal is sampled more than twice its frequency and the signal is represented better in the digital domain.

After converting the analog signal to a digital signal by ADC, a downconversion can be required by the General Purpose Processor (GPP). For example, when analog data is sampled at 600MS/s, the amount of data is too high for most GPPs, thus data rate conversion may be performed before transferring digital data to GPP. The down-converted digital signal is then sent to GPP to perform digital signal processing. In some SDRs, FPGA is used to make digital signal processing for computationally intensive tasks.

2.2. Physical Layer Structure of 802.11 Signals

Wi-fi technology helps devices to connect to the internet using radio waves. Wi-Fi devices are Wireless Local Area Network (WLAN) products that employ the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards. In our work, we have classified 802.11b/g/n signals which are emitted by WLAN devices.

Protocol signals use frequencies of 2.4 and 5 GHz, which are located in the ISM band that is reserved internationally to be used for industrial, scientific and medical purposes rather than telecommunication. The 802.11 standard defines the physical (PHY) and media access control (MAC) layer of the OSI model. 802.11 devices also apply carrier-sense multiple access with collision avoidance (CSMA-CA) method, which listens to the shared spectrum before sending any packet to prevent collisions. 802.11 signals are located in different channels in the spectrum with 22 MHz bandwidth and 5 MHz intervals between center frequencies. Thus, WLAN channels overlap with each other resulting in an interference problem. As depicted in Figure 2.3. However, most devices use channels 1, 6 and 11 due to their non-overlapping locations.

In our work, 802.11b/g/n signals are classified using physical layer features such as modulation and radio frequency bandwidth. In Table 2.1, some of the physical layer features of 802.11b/g/n signals are shown. In the following subsections, the physical

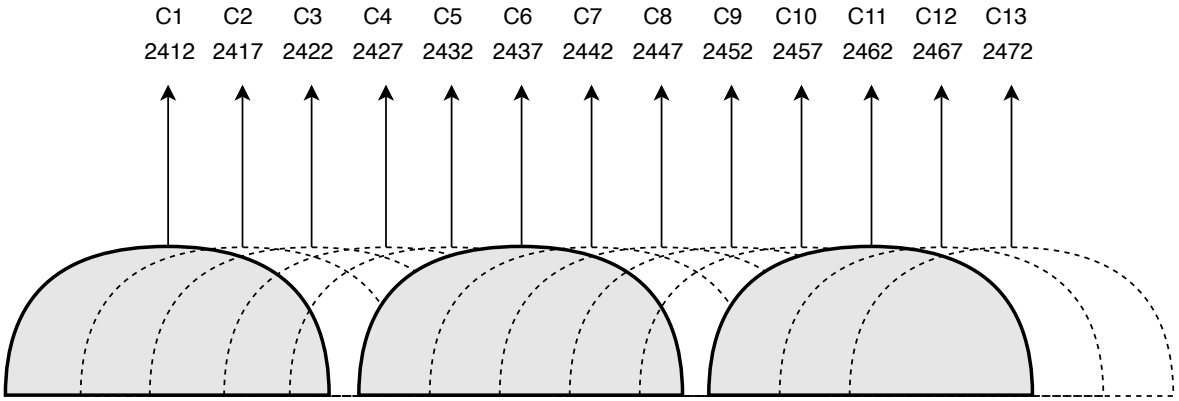


Figure 2.3. WLAN Channels in Frequency Spectrum.

layer details of 802.11b/g/n signals will be explained to show distinguishable features of these signals. In our designed CNN, frequency domain features of signals will be used, thus only frequency domain details will be given related to 802.11 signals.

Table 2.1. Physical Layer Features of 802.11b/g/n Signals.

Standard	**Frequency**	**Bandwidth**	**Phy**	**Max Data Rate**
802.11b	2.4 GHz	22 MHz	DSSS	11 Mbit/s
802.11g	2.4 GHz	20 MHz	DSSS/OFDM	54 Mbit/s
802.11n	2.4/5 GHz	20/40 MHz	MIMO-OFDM	600 Mbit/s

2.2.1. 802.11b

802.11b employs the Direct Sequence Spread Spectrum (DSSS) technique for transmission of signals. It occupies a bandwidth of 22 MHz. In DSSS, the signal is transmitted using thousands of different carriers resulting in a larger bandwidth than original signal content. Due to the fact that the signal is spread to large bandwidth, the power of the signal is reduced considerably which buries the signal in the noise. Because of this, DSSS provides better security and robustness to interference.

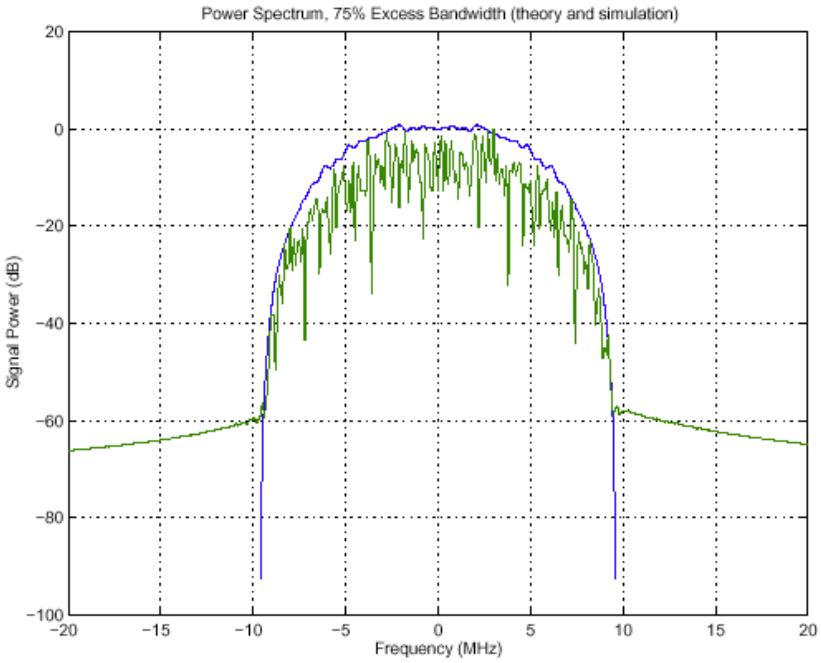


Figure 2.4. Typical IEEE 802.11b Signal [2].

The data signal is modulated by a known bit sequence which is called *Pseudo Noise* (PN). PN is a high bandwidth signal that scrambles and spreads data to large bandwidth. Here, the period of PN pulse is also called *chip duration*. Frequency domain representation of typical IEEE 802.11b signal is shown in Figure 2.4.

2.2.2. 802.11g/n

IEEE 802.11g standard employs an OFDM transmission scheme, which increases the maximum data rate up to 54 Mbit/s from 11 Mbit/s, which is obtained by the 802.11b standard that uses DSSS technology. 802.11g is a Single Input Single Output system.

OFDM is a multicarrier transmission technology that employs many subcarriers with a lower data rate to transfer data. An important reason to use OFDM is increasing the resistance towards frequency selective fading and narrowband interference. When interference is available in the channel, only small parts of subcarriers will be affected by interference due to the multicarrier nature of OFDM. The data corrupted by in-

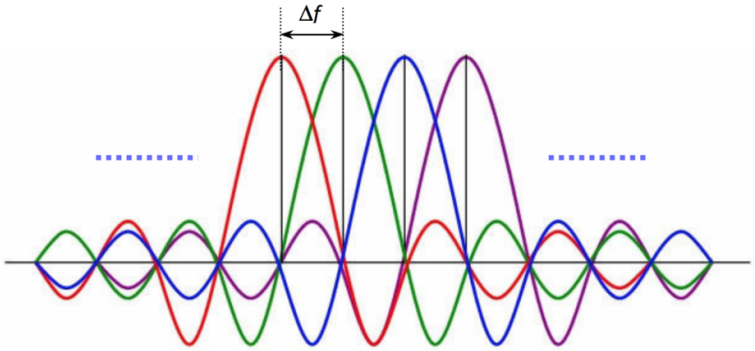


Figure 2.5. Frequency spectrum of OFDM subcarriers [3].

terference can be corrected by error coding techniques. In OFDM, subcarriers are arranged as orthogonal to each other to use the frequency spectrum efficiently and to increase the data rate. The orthogonality of subcarriers also prevents Inter-Carrier-Interference (ICI), which leads to power leakage between subcarriers [63]. In Figure 2.5, OFDM subcarriers are shown. It can be observed that individual subcarriers are orthogonal to each other and signal can be demodulated without error. If each subcarrier is modulated with a good modulation scheme, the data rate can be increased significantly.

802.11g standard uses 20 MHz channel bandwidth and 16.6 MHz occupied bandwidth. A single OFDM symbol includes 64 subcarriers 48 of these subcarriers are data, 4 of them are pilot carriers and the remaining are null subcarriers. Typical 802.11g OFDM symbol is shown in Figure 2.6. Data carriers are used for the message signal. Pilot carriers are pre-defined subcarriers with a known modulation scheme that are used by the receiver side to correct frequency and phase offset.

802.11n, which is introduced in 2009 uses OFDM modulation with 52 data subcarriers. This provides an 8% enhancement compared to the previous standard 802.11g. 40 MHz channels with 128 subcarriers, basically double the data rate, are defined for this standard. Also, multiple-input-multiple-output (MIMO) technology and some modulation and coding techniques are added. 802.11n physical layer can be configured

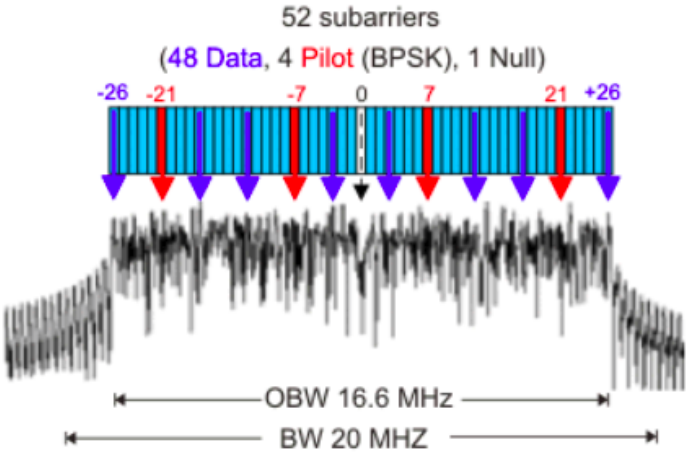


Figure 2.6. OFDM signal of 802.11g standard [4].

using different Modulation and Coding Schemes (MCS). Data rate can increase up to 600 Mbit/s with these improvements [64].

Table 2.2. Physical Layer Features Comparison of 802.11g and 802.11n.

Feature	**802.11g**	**802.11n**
Max Data Rate (Mbps)	54	600
Bandwith (MHz)	20	20/40
FFT Size	64	64/128
Data-Pilot Carriers	48,4	52,4/108,6
Antenna Tech	SISO	MIMO
Modulation	BPSK, QPSK, 16-QAM, 64-QAM	
Subcarrier Interval (kHz)	312.5	

In Table 2.2, physical layer features of 802.11g and 802.11n are compared. It can be observed that although 802.11n is more advanced technology, they use the same modulation techniques, which is OFDM. However, there is a slight difference between their modulation schemes. For 20 MHz channel, 802.11n employs 56 data and pilot subcarriers, but 802.11g uses 52 subcarriers for data and pilots. Thus, there will

be a difference in their occupied bandwidths. In equation 2.1 and 2.2, the occupied bandwidths of 802.11g and 802.11n are calculated for 20 MHz bandwidth. There is also one DC subcarrier in the middle of the OFDM symbol. This occupied bandwidth difference will help our CNN algorithm to differentiate these signals from each other for 20 MHz.

$$BW_{occ_g} = \Delta_f n = 312.5\,kHz * 53 = 16.57\,MHz \tag{2.1}$$

$$BW_{occ_n} = \Delta_f n = 312.5\,kHz * 57 = 17.8\,MHz \tag{2.2}$$

3. IMPLEMENTATION HARDWARE AND SOFTWARE TOOLS

In proposed work, an SDR platform that is called Universal Software Radio Peripheral (USRP) is employed as hardware to examine created the wireless signal classifier. In this section, the detailed architecture of the USRP platform will be presented. Also, basic software libraries and tools such as UHD, RFNoC and GNU Radio that used to control USRP will be expressed in the following sections.

First of all, section 3.1 will explain the hardware architecture of the USRP E310 platform. In section 3.2, UHD, RFNoC and GNU Radio will be explained. UHD runs on a general purpose processor and controls the USRP device. RFNoC is a network-distributed heterogeneous processing tool with a focus on enabling FPGA processing in USRP devices [65]. GNU Radio is an open source general signal processing library that provides basic signal processing blocks such as filters, modulators, resamplers, math operators, measurement instruments, etc [66]. These software tools will be explained in more detail in the following sections.

3.1. Hardware: USRP E310

Accessing the RF spectrum using SDR was not easy due to the high cost of hardware. To bring a solution to this problem, the USRP project was started by Matt Ettus by aiming low-cost SDR for researchers and students in 2003. The first USRP device is released in 2005 by Ettus Research. USRP device hardware is open source and schematics can be accessed from the internet. Also, all the USRP devices use an open source software driver, which is called Universal Hardware Driver (UHD). General USRP hardware architecture is shown in Figure 3.1. Basically, USRP devices are created by three hardware parts, which are GPP, USRP motherboards, and RF daughterboards. A computer with GPP is required to interface with USRP devices. UHD runs on a GPP as a bridge between the USRP device and computer to transfer digital samples. Digital signal processing can be achieved on digital data using other libraries or tools such as GNU Radio, MATLAB, etc.

The motherboard runs as an interface between a digital baseband signal and an analog signal. The computer can be connected to the motherboard using high-speed ethernet, USB or PCI. Incoming IF or baseband analog signal is sampled by ADC on the motherboard and send to the FPGA. Also, timing, control, and digital down/upconversion are performed in FPGA on motherboard [26].

The RF daughterboard is used to convert between RF signals and analog IF / baseband signals. This conversion is employed using the mixers, filters, oscillators and amplifiers etc. Also, analog bandwidth of device depends on the RF daughterboard [5]. Analog bandwidth is amount of bandwidth that is declared by low pass filters in daughterboard. These filters are placed before ADC/DAC conversion, which can be observed in Figure 3.1. USRP devices have many categories. The main categories are:

- Bus (B) Series: Connected to host computer via USB connection
- Network (N) Series: Connected to host computer via Ethernet connection
- High Performance (X) Series: Connected to host computer via Ethernet or PCI-Express connection

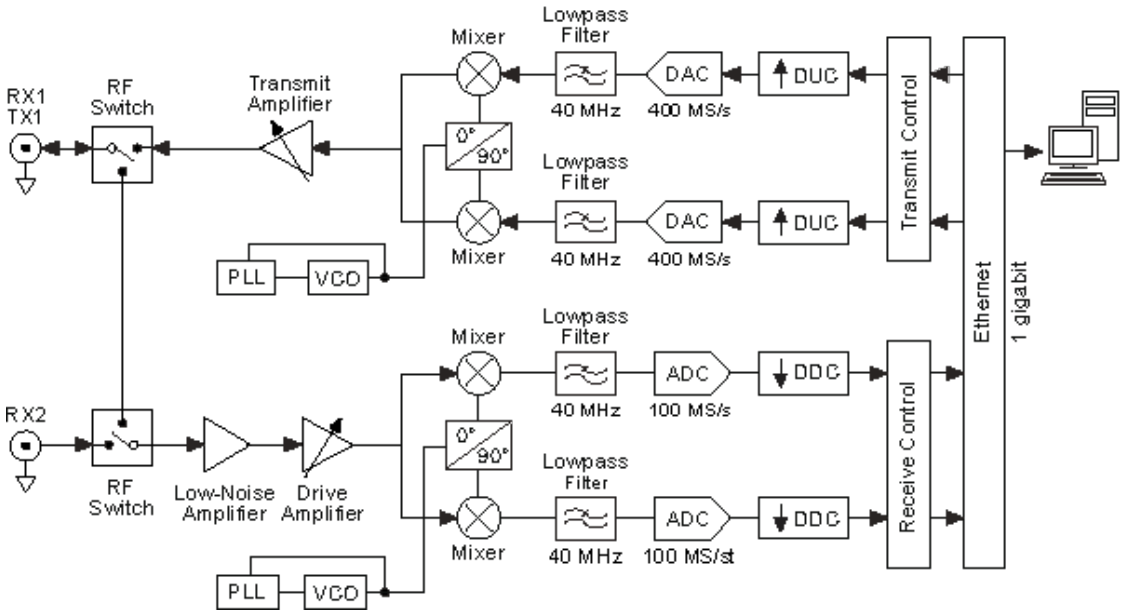


Figure 3.1. Architecture of USRP Devices [5].

- Embedded (E) Series: Devices have embedded processor to run as standalone device

Bus, Network and High-performance series of USRP solutions are connected to host computers and used as slave devices. Host computer configures the FPGA in SDR and uses it to receive and transmit RF signals. Most computationally intensive tasks such as downconversion, upconversion, decimation, interpolation, etc. are employed in FPGA. Tasks that require less computation are done in GPP. Most of the time, the sample rate is decreased through GPP to facilitate signal processing. For example, although the SDR can handle 200 MS/s sample rate, the GPP can only process up to 8 MS/s. So, the FPGA will make downconversion on incoming samples by benefiting software and the GPP will handle downsampled data properly. These types of devices can receive and transmit very high bandwidth of signals depending on the size of FPGA.

USRP E310 is one of the devices from the Embedded series of USRP. This means E310 includes an embedded processor as well as FPGA to operate as a standalone

Figure 3.2. USRP E310 [6].

device without connecting to any host computer. This feature and its small size make USRP E310 very suitable to deploy in field applications. The USRP E310 is shown in Figure 3.2.

USRP E310 is a 2x2 MIMO device and can span the frequencies between 70MHz - 6GHz using filter banks in the front-end. It has instantaneous bandwidth up to 56MHz thanks to AD9361[8]. AD9361 is an ADC/DAC from Analog Devices that has an output data rate of up to 61.44MHz. Although AD9361 can generate samples faster, the output data rate is acquired using decimators in AD9361 after sampling analog data. Hardware architecture of USRP E310 is shown in Figure 3.3.

Xilinx Zynq 7020 SOC is used as a baseband processor. It includes a dual-core ARM processor and FPGA to accelerate IF and baseband processing. It uses the Open-embedded platform for creating an efficient custom operating system. Openembedded is a build system that provides different layers for constructing an operating system with requirements of the application. In E310, the board support package layer of E310, Xilinx and also SDR layer are added to the operating system.

E310 can be connected to a network or directly to a computer via ethernet. Users can develop their SDR applications in computer. When it comes to testing, applications

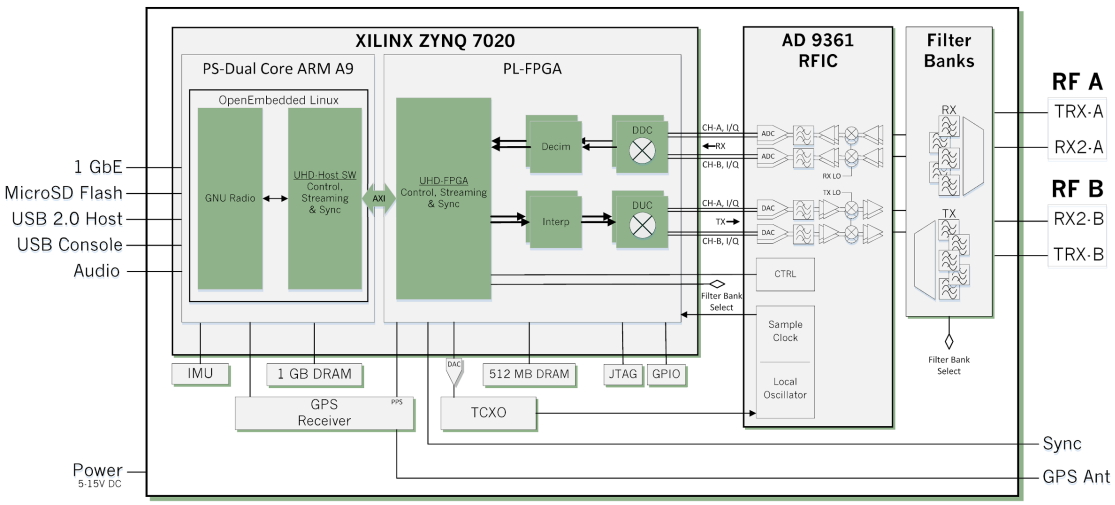


Figure 3.3. Hardware Architecture of USRP E310 [6].

are transferred to E310 and run in embedded processor. Also, for debugging purpose, the data from USRP to host computer can be transferred via network using UDP packets. This way, processed data can be observed in real time using GNU Radio software on host computer.

Basic receiver data flow in E310 is such that RF signal is first collected by antennas and filtered using filter banks according to the selected center frequency. Then analog data is sampled in the AD9361 IC and converted to digital data. Here, analog data is converted to IQ data which includes in-phase and quadrature parts of complex RF signal. The ADC output data rate can be up to 61.44MHz. If the user requested a lower data rate, data is down-converted to the requested data rate in FPGA using DDC. Finally, digital IQ samples are passed thorough a baseband ARM processor to make signal processing. This is the basic flow of the USRP E310 RF signal reception. In this receiver scenario, FPGA is not configured by the user, it is automatically configured by USRP software. If the user needs more specific computational tasks such as Fast Fourier Transform (FFT), FPGA can be programmed to take FFT of digital samples. In this thesis, USRP E310 is programmed using UHD, RFNoC and GNU Radio software libraries. These are explained in the following subsections.

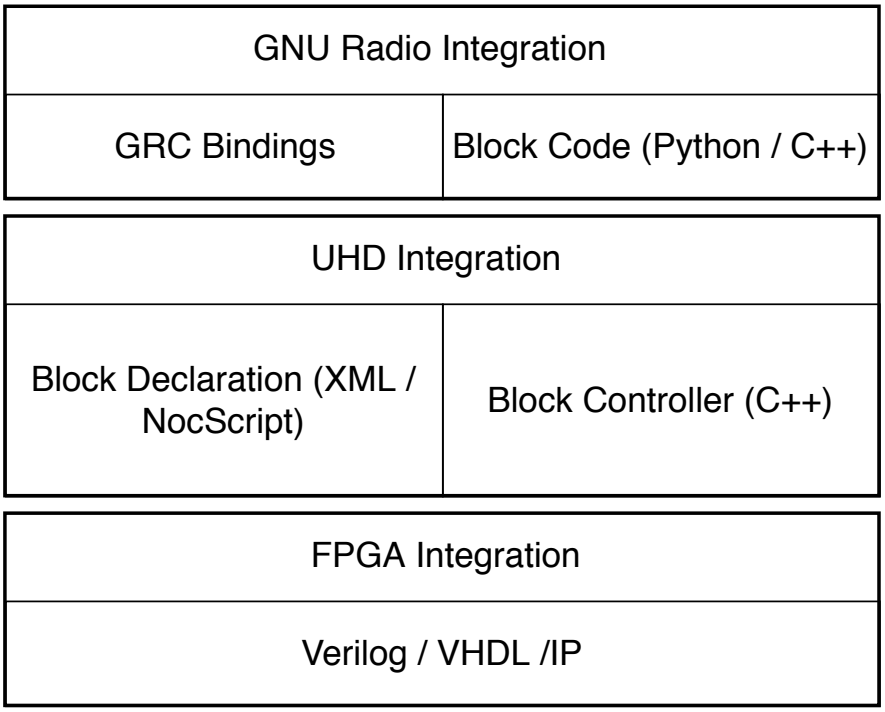


Figure 3.4. Software Stack of USRP E310.

3.2. Software

As it is mentioned before, SDR is a fully configurable device that can be configured using a Hardware Description Language (HDL) and other general-purpose software languages such as C, C++, Python. There are several software stacks to configure USRP devices. Software stacks to control USRP E310 is shown in Figure 3.4. For most hardware-specific parts, there is a Verilog language to configure the FPGA. UHD is positioned on top of the FPGA Integration stack. UHD runs on GPP and provides control over FPGA and data flow that goes from and to GPP. On top of the UHD Integration stack, there is an open-source general-purpose signal processing library which is called GNU Radio. GNU Radio provides many signal processing blocks to the user such as filters, demodulators, math operation blocks, etc. Users can also create their own GNU Radio blocks. These libraries will be examined in the following sub-sections.

3.2.1. Universal Hardware Driver

The main purpose of SDR is configuring and controlling hardware using user-defined software applications. First-generation USRP devices are controlled via software API that is specific to the device. However, this approach is not scalable and APIs should be changed for different device types. UHD is designed to solve these problems by controlling all USRP devices using the same driver. UHD is an open-source library that can run on Linux, Windows, MAC OS and also Embedded Linux. It provides an API to control device-specific features by abstracting away low-level details of hardware [26].

UHD helps the user control the transfer of digital samples from and to the USRP device by means of several parameters such as sample rate, center frequency, analog gain, data type, etc. UHD driver is written in C/C++ languages. FPGA code is written in Verilog and the configuration of FPGA is controlled by the C/C++ code of UHD. Other third-party libraries can be built on the top level of UHD APIs to control USRP devices. This is expressed in Figure 3.5. As an example, MATLAB, GNU Radio, LabVIEW and others are built on the top level of UHD [67].

In host computer-based USRPs, the motherboard of USRP is connected to a computer via ethernet, USB or PCI-Express since UHD is installed on the computer. However, in embedded USRP series, UHD should be cross-compiled and installed on the embedded processor. For the simplicity of application development, UHD is also installed on another computer that includes GPP since the embedded processor is computationally limited compared to GPP. As a result, users can design their UHD application on GPP based computer and run the application on an embedded processor for testing purposes.

3.2.2. RF Network on Chip

Processing units such as CPU and FPGA are improving roughly with Moore's Law. This provides engineers to design more complex and capable devices. Designing

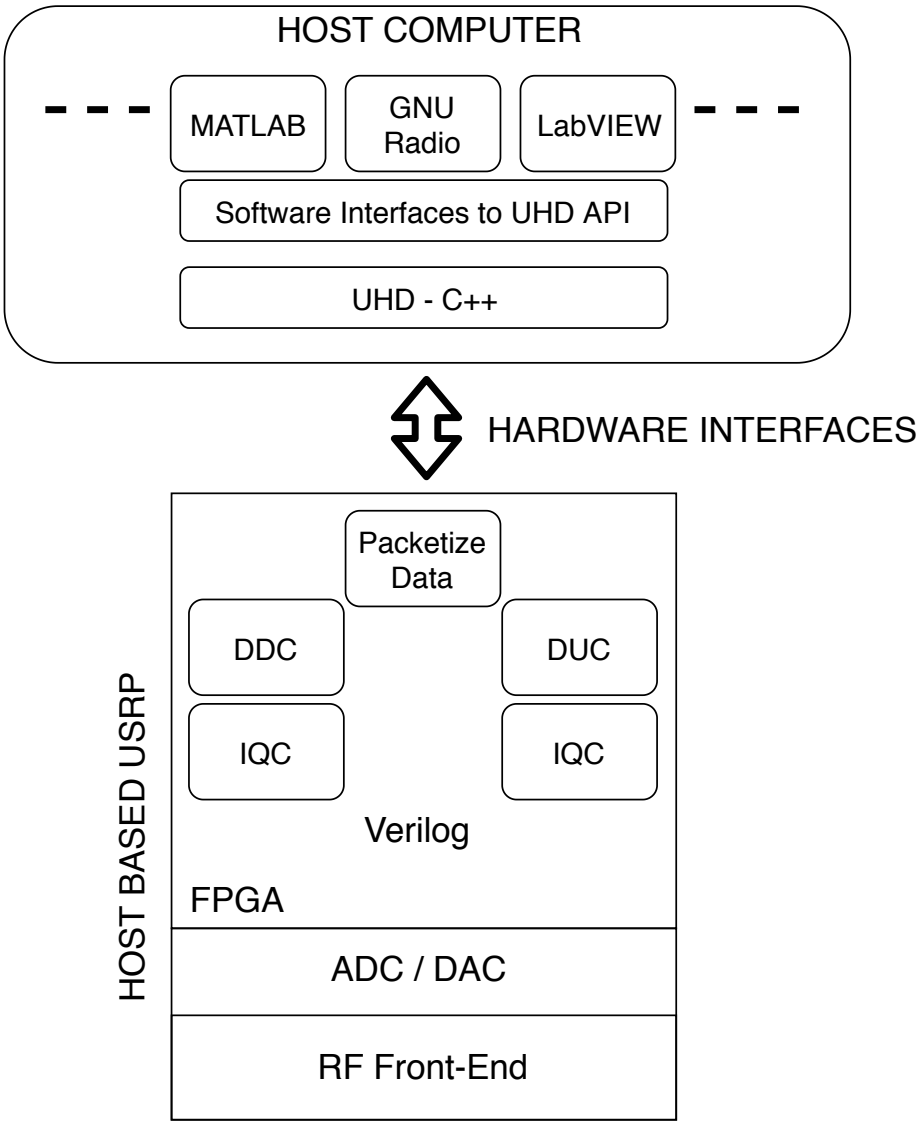


Figure 3.5. Some of the Software Tools Interfaces with UHD.

more powerful systems also increases the complexity of these systems. But human resource ability does not increase with the same trend. The gap between human abilities and emerging systems results in a design crisis [26].

In software, this difference has been closed using advanced design tools. But in the FPGA domain, although some design tools and standards facilitate the design, code re-usability problems continue to exist. RFNoC is designed to overcome communication protocol design effort between Intellectual Properties (IP). RFNoC protocol is a design process similar to placing simple blocks like legos to create complex systems. The

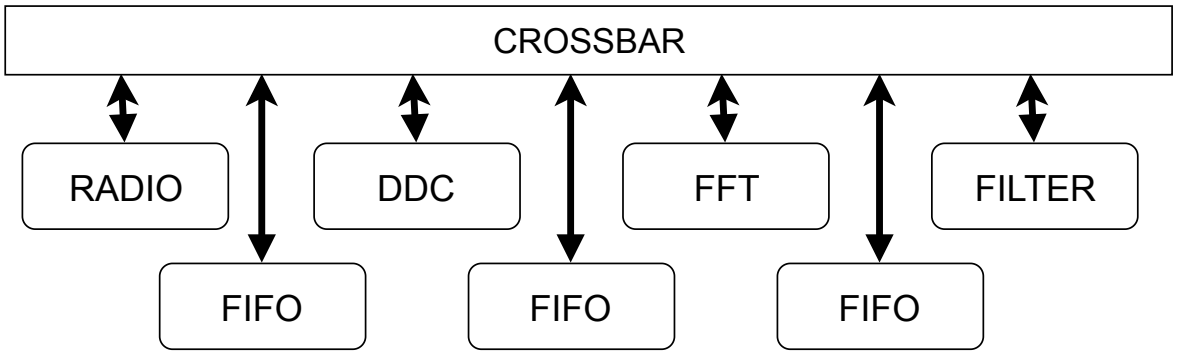


Figure 3.6. RFNoC CE Communication Interface.

legos, in this case, are called as Computation Engine (CE), which are also called as RFNoC blocks. RFNoC is a distributed network of CEs and radio front-ends (radio) that facilitates data flow on and off of an FPGA in a transparent way [26]. Therefore, engineers can focus their efforts on designing algorithms rather than designing FPGA interfaces. A sample RFNoC network is shown in Figure 3.6.

RFNoC infrastructure allows establishing small and large scale real inter-chip and intra-chip networks, which means RFNoC can be easily scaled. RFNoC blocks communicate with each other asynchronously. There is no master-slave system in RFNoC, thus any block can control other blocks by sending a control packet with proper format.

CE is a small hardware block in FPGA that plays a specific role by processing the incoming stream from other CEs and producing output stream to other CEs. Filters, FFT blocks, modulators, etc are some of CE examples. User has no obligation to design CEs for a simple task; it can be either designed for more complex tasks such as an OFDM demodulator that includes synchronizer, equalizer, decoder, etc. or each of these tasks can be designed as independent CEs. The latter is preferable because of code re-usability. An FIR filter can be used in any task in signal processing applications; thus, designing FIR filter as RFNoC block is a smart choice.

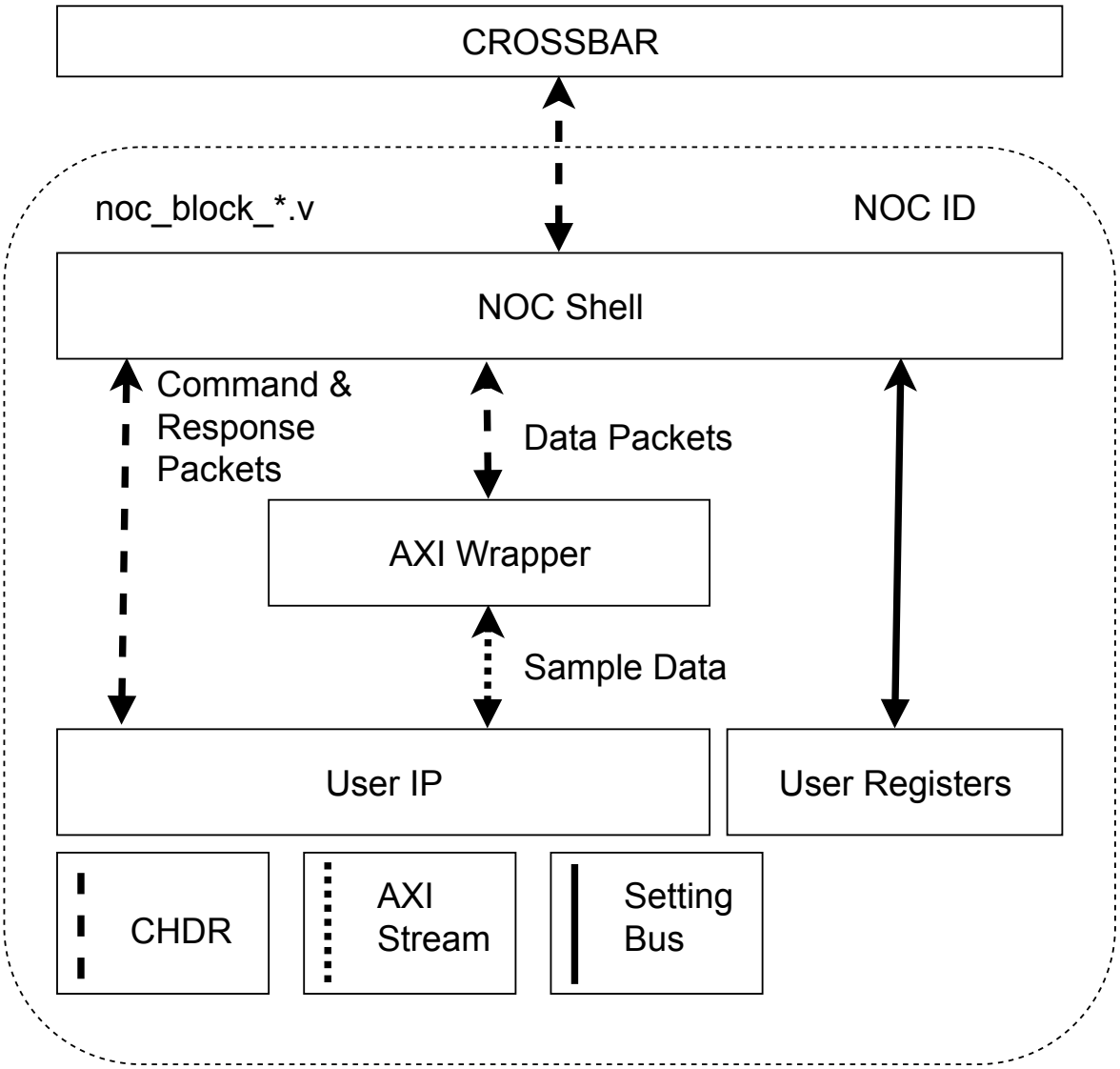


Figure 3.7. RFNoC Block Architecture.

There is a mandatory block in RFNoC named as Radio Block, which establishes a digital interface with radio hardware. Every task that is related to radio hardware is realized in this block such as controlling gain, baseband sampling rate and other digital domain-related tasks such as filtering, decimating, timestamping.

NOC shell is provided to CEs to communicate with the remaining RFNoC blocks. NOC Shell performs core tasks such as packet muxing and demuxing, flow control and register bus settings [65]. A sample RFNoC block and its interface is depicted in Figure 3.7. NOC Shell expects a special packet, which is called Compressed Header (CHDR).

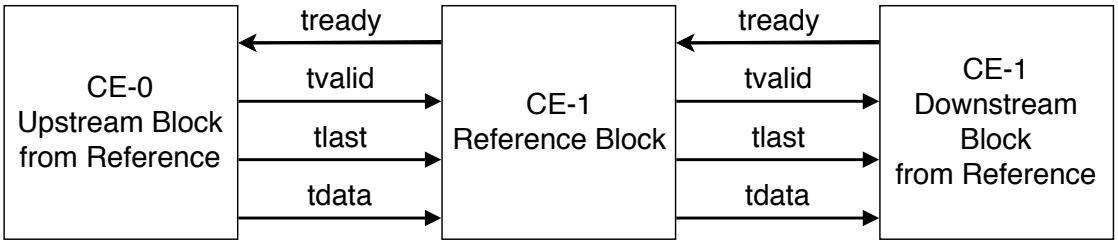


Figure 3.8. RFNoC CE Interfaces with Other CEs.

RFNoC provides interface wrapper, which encapsulates existing or external IP to use with RFNoC. Interface wrapper converts Advanced Extensible Interface (AXI) Stream data to CHDR packets, which, means the user can use existing Xilinx IP blocks with RFNoC by using AXI wrapper interface.

Each block includes four AXI Stream interfaces (tready, tvalid, tlast, tdata) and a setting register bus. CEs exchange data between each other using these signals. The structure of this system is shown in Figure 3.8. Each CE is independent of each other and can be designed using several tools including Verilog, VHDL, and Xilinx Vivado HLS. After designing RFNoC block, blocks are placed on the FPGA image. Beginning with third-generation USRP devices, all USRPs will support RFNoC within UHD.

3.2.3. GNU Radio

Although there are several options in SDR development, the open-source GNU Radio is one of the most popular. GNU Radio provides blocks such as filters, re-samplers, decoders, etc. for designing different signal processing applications. It also provides a software development environment to create custom blocks that are called Out of Tree Modules.

GNU Radio contains various block types such as source, sink, sync, decimator, interpolator, general, tagged stream. GNU Radio blocks are connected to each other like a signal processing chain. One block produces samples while another block consumes previous block's samples. GNU Radio employs a dynamic scheduler that passes chunks

of samples between blocks. It increases efficiency due to moving samples by chunks instead of individual samples. This makes GNU Radio more suitable for real-time signal processing using streams.

GNU Radio employs Python and C++ languages. All GNU Radio blocks that perform signal processing tasks are realized using C++. There is also a graphical tool called GNU Radio Companion (GRC), which allows connecting GNU Radio blocks using a graphical interface and generates necessary Python codes to run GNU Radio applications. Basic GNU Radio development flow is as follows: users design their blocks in C++ and connect blocks using graphical interface GRC for avoiding unnecessary coding for connections.

GNU Radio can also be used with SDRs by controlling them using their drivers. In our work, we will use GNU Radio with USRP E310 to design applications. As it is shown in Figure 3.4, GNU Radio can interface with UHD and RFNoC. Some wrapper codes called as gr-uhd and gr-ettus for UHD and RFNoC, respectively, are written by Ettus Research in order to use GNU Radio with RFNoC and UHD. Gr-uhd and gr-ettus provide functions to control USRP hardware using GNU Radio.

User can design custom RFNoC blocks in FPGA using Verilog to speed up applications by utilizing hardware parallelism. UHD and GNU Radio wrapper codes should also be written to use RFNoC blocks with GNU Radio.

4. REAL-TIME 802.11b/g/n SIGNAL CLASSIFIER (BGN-C)

Real-Time detection of RF signal is a key point for many applications such as spectrum monitoring, dynamic spectrum access, defense applications, etc. The signal detection and classification field attract the attention of machine learning scientists for several years. Some authors classified RF signals using SVM [16], CNN [68], Residual Neural Networks [22] and Neural Networks [69], etc. In our design, 802.11b/g/n signals are classified in real-time using CNN based method. Generally, signal classification tasks are performed using high-end devices that are capable of high sampling rates and computational capability. In this work, the classification of signals is employed on an embedded device that has limited RF bandwidth and computational power.

In this section, the implementation details of Real-Time 802.11b/g/n Signal Classifier are presented. First of all, the general system architecture will be given with rough details. In the following, the DAE is provided. In the next section, the preprocessor is given. In the last sections, the implementation details of CNN are explained.

4.1. General System Architecture

In the BGN-C system several logical blocks namely DAE, preprocessor and signal classifier, are designed to realize proposed real-time classification hardware. The proposed system works in a 2.4 GHz ISM band to classify 802.11b/g/n signals. The device can receive signals that have a bandwidth of up to 56 MHz that covers 802.11b/g/n signals. General system architecture block diagram is shown in Figure 4.1.

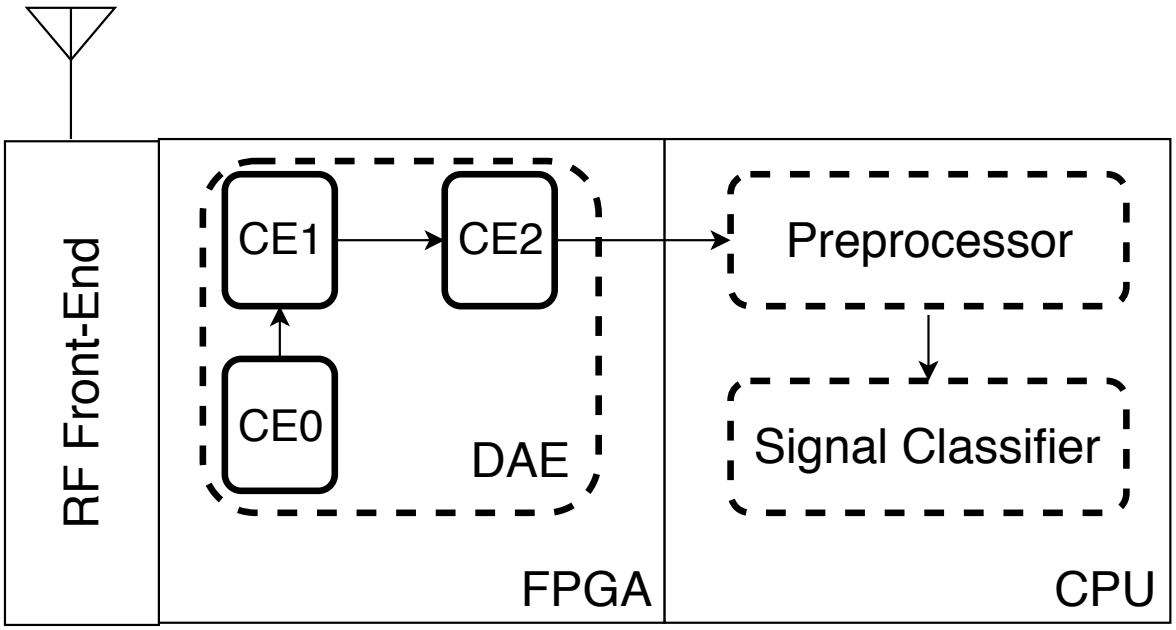


Figure 4.1. General System Architecture of BGN-C.

BGN-C is realized in USRP E310 hardware, which has embedded CPU and FPGA. RF analog signal is collected using RF Front-end, which has selectable analog filters (filter bank) and gain control [70].

Analog signal that passed through filter banks is sent to AD9361 [71] IC, which is located in E310, to make an analog-to-digital conversion. AD9361 has a direct conversion receiver that consists of a Low Noise Amplifier (LNA), IQ amplifier, mixer, and band shaping filter. AD9361 down-converts the signal to baseband using LOs. AD9361 includes 12-bit ADC with an adjustable sampling rate for converting the incoming analog signal to digital. It also includes a decimation filter and a 128-tap FIR

filter for converting a high sampling rate digital signal to a suitable baseband sampling rate that is up to 61.44 MHz. This is the sampling rate of IQ samples, which pass to FPGA.

After samples are fed to FPGA from AD9361, DAE takes raw IQ samples and apply several signal processing steps to obtain meaningful features of the incoming signal. It also applies down-conversion to decrease the sampling rate in order to transfer digital samples to the CPU.

The sampling rate for CPU can handle changes according to employed application. Generally, FPGA should perform down-conversion to incoming samples and transfer the samples to CPU with up to 8 MS/s depending on application load. If samples are sent with a higher sampling rate, the CPU cannot consume samples and overflow occurs. Overflow usually does not crash the program, but some samples will vanish. Incoming digital samples to CPU are handled by the preprocessor to generate spectrogram images for the signal classifier. After this step, the signal classifier passes samples through CNN to make a prediction of the input signal.

DAE, preprocessor and signal classifier are explained in detail in the following sections.

4.2. Data Acquisition Engine (DAE)

DAE is a high-speed signal processing module that runs on FPGA with a sampling rate of 40 MS/s. DAE gets raw IQ samples from the radio and takes FFT of signal and smooths FFT outputs by passing them from a simple IIR filter. Down-conversion is necessary to decrease the sampling rate because the CPU cannot handle the high data rate due to its limited computational capability. Finally, smoothed FFT outputs are written to a file for a pre-processing step. It was observed that the classification accuracy of CNN improves when FFT outputs are smoothed. Features of DAE is shown in Table 4.1.

Table 4.1. Features of DAE.

Feature	**Value**
Sampling Rate (FPGA)	40 MHz
Sampling Rate (CPU)	2.3 MHz
Analog Bandwidth	56 MHz
FFT Size	512
Center Frequency	2440 MHz

All tasks in DAE are programmed using GRC and RFNoC. In Figure 4.2, GNU Radio flowgraph of DAE is shown. These blocks are provided by Ettus Research and they are used to design DAE in USRP E310. There are several steps to prepare this flowgraph for running GNU Radio on USRP E310, which are listed below:

- Firstly, the required RFNoC blocks should be defined for the application. These blocks should be designed on FPGA using Verilog unless Ettus Research does not provide them.
- New FPGA image should be generated using designed RFNoC blocks. The output .bit file will run on the FPGA side.
- UHD and RFNoC should be compiled for E310 to control FPGA from the CPU.
- GNU Radio should also be compiled for the usage of GNU Radio on E310.
- gr-ettus module that is provided by Ettus Research should be compiled for GNU Radio - RFNoC integration.
- All these compilation steps should be performed for the ARM processor that runs on E310.

All of these compilation and FPGA image generation details will not be given in the thesis report due to unnecessary details.

DAE can be separated into three main parts as radio interface, frequency domain representation of Signal and smoothing and downsampling. Detailed explanations for

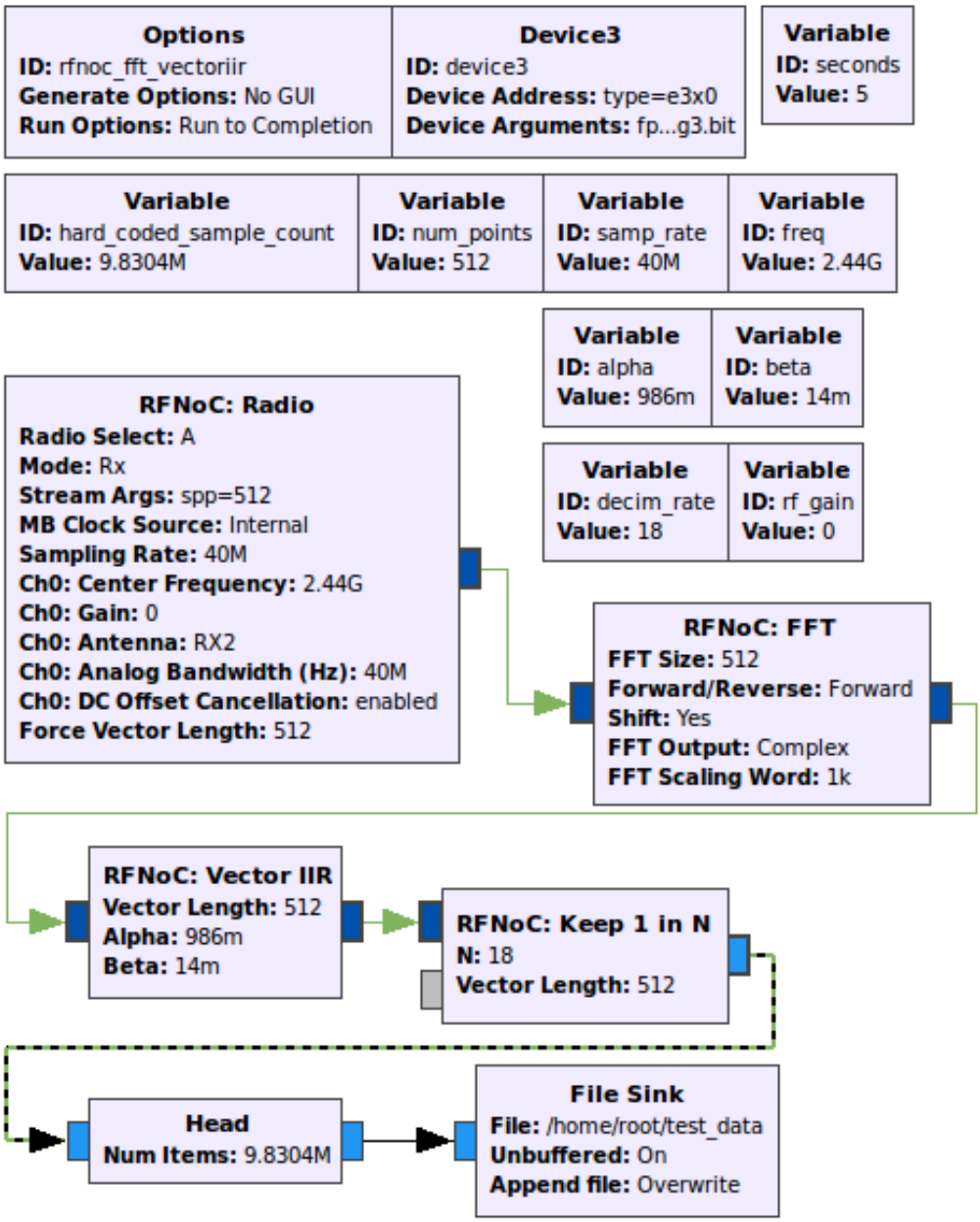


Figure 4.2. GRC Flowgraphof DAE.

these parts are given in the following subsections.

4.2.1. Radio Interface

Actual radio hardware is controlled by RFNoC radio block, which sets clock source, sampling rate, center frequency, analog gain, analog bandwidth of reception

filters, etc. All tasks related to radio hardware are employed in this block. As mentioned before, analog samples are received by AD9361 transceiver and amplified by the LNA in it. Analog signal is mixed, filtered and amplified to be converted to baseband signal before digitization by ADCs. Here, I (In-phase) and Q (Quadrature) signals are generated using LO and mixer according to (4.1). Basic IQ demodulator is shown in Figure 4.3. IQ signals are directly converted to baseband due to the direct conversion

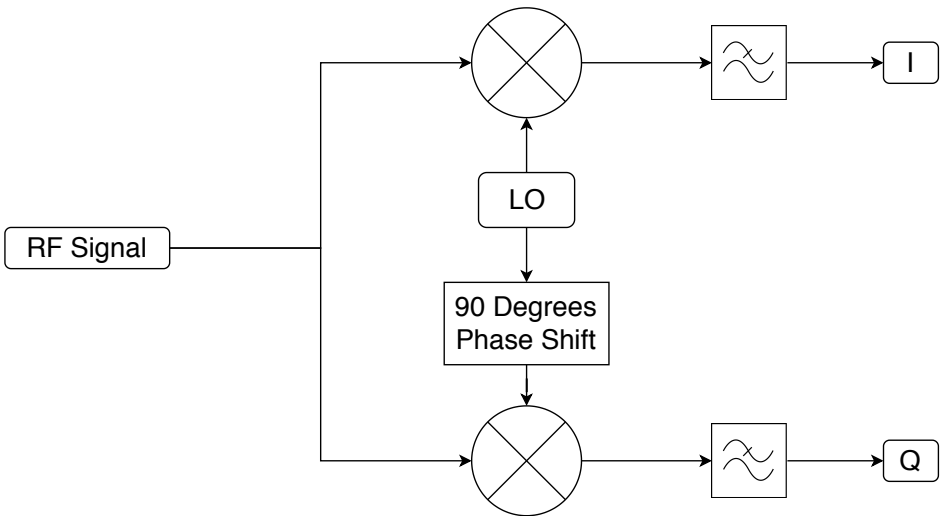


Figure 4.3. Basic IQ Demodulator.

receiver architecture of AD9361. RF receiver path of AD9361 contains low pass filters up to 56 MHz cut-off frequency, 12-bit ADC and digital decimation filters [72]. Analog IQ samples are sampled by ADC at high speed and down-converted using decimation filters to the desired data rate to send samples to FPGA. The data rate from ADC to FPGA is up to 61.44 MS/s. In AD9361, the FPGA data rate is 40 MS/s and analog bandwidth is selected as 56 MHz, which is more than enough to capture 802.11 signals that have 20 MHz bandwidth.

$$I = A\cos(2\pi ft), \qquad Q = A\sin(2\pi ft) \tag{4.1}$$

As depicted in Figure 4.3, LO signal is mixed with input directly to generate I signal; 90 degrees phase shifted version of the LO signal is used for generating Q signal. In practice, the delay is not exactly 90 degrees, which introduces an error called IQ

Imbalance and also another error called DC Bias. These errors are also corrected within AD9361. This behavior results from the nature of direct conversion receivers [73].

Next, the samples are transferred to FPGA and in particular to RFNoC radio block. AD9361 is also configured by radio block to set proper analog gain, analog bandwidth, and baseband sampling rate. Radio block adds a timestamp to incoming samples from AD9361 and sends samples to the RFNoC network. There is also a parameter called spp, which provides more samples in one packet, in the radio block. Increasing this value decreases overall overhead on the network.

4.2.2. Frequency Domain Representation of Signal

Frequency domain analysis demonstrates how the energy of the signal is distributed to different frequency bands. However, time-domain analysis shows how the signal changes over time. Commonly, frequency domain analysis is employed to identify the characteristics of the signal.

In our situation, 802.11b/g/n signals should be analyzed. Both time and frequency domain can provide information that can be used in a neural network to classify these signals. Time-domain can give information about signal duration. But signal duration can be different for separate transmissions due to the content of each packet. So, this information is not very useful. Frequency domain, on the other hand, can give center frequency, bandwidth, frequency content, modulation type, phase shift information about the desired signal. Therefore, frequency domain transformation will be employed to obtain the frequency characteristics of the signal.

In frequency domain transformation, Short Time Fourier Transform (STFT) is taken for 512 points using time-domain samples. STFT is employed to find the frequency content of shorter segments. Generally, the STFT of these shorter segments is plotted side by side to visualize the frequency content of signal changing by time. Here, FFT, which is an efficient Fourier Transform algorithm, is used to take STFT.

$$X(n,k) = \sum_{m=0}^{L-1} x[m]\omega[m-n]e^{-j\frac{2\pi}{N}m} \tag{4.2}$$

In equation (4.2), STFT of discrete signal $x[m]$ is defined as $X(n,k)$, where N is the total number of samples, L is the length of window, $\omega[m]$ is window function, which is used to decrease spectral leakage from the edges of the spectrum due to non-periodicity of samples.

In our implementation, segments are chosen as 512 points of length and rectangular window function is used. After performing STFT operation, generated vectors are combined to create spectrograms.

4.2.3. Post-Processing the FFT Outputs

In this section post-processing RFNoC blocks, which are *vector_iir* and *Keep 1 in N*, will be explained in detail. Both blocks run in FPGA.

4.2.3.1. Single-Pole IIR Filter. Raw digital IQ samples passed through the FFT block, which performs STFT for 512 data samples. By doing this, the frequency components of a short segment are extracted. FFT outputs are then fed into Single-Pole IIR Low Pass Filter, which is a kind of moving average filter, to smooth FFT outputs. The most significant feature of this filter is its simplicity, which is crucial for our limited hardware. FPGA part of USRP E310 is relatively small, therefore more than 6 RFNoC blocks cannot be fit inside the FPGA (Actually if the size of RFNoC blocks are larger, this number will reduce).*vector_iir* block, which is FPGA implementation of Single-Pole IIR Filter, remarkably decreases noise level using the smoothing effect.

Single-Pole IIR Filter can be defined by equation (4.3), which has a single parameter β smoothing factor between 0 and 1. $x[n]$ and $y[n]$ are input and output samples

of the filter, respectively. The smoothing process reduces the noise level without using system resources too much.

$$y[n] = \beta x[n] + (1-\beta)y[n-1], \qquad 0 < \beta < 1 \tag{4.3}$$

Transfer function of the filter can be found after Z-Transform as shown in (4.5).

$$Y(z) = (1-\beta)z^{-1}Y(z) + \beta X(z) \tag{4.4}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\beta}{1-(1-\beta)z^{-1}} \tag{4.5}$$

$-3\ dB$ cut-off frequency ω_c can be found by solving equation (4.6).

$$\left|H(z = e^{j\omega_{3dB}})\right|^2 = \frac{1}{2} = \left|\frac{\beta}{1-(1-\beta)e^{-j\omega_{3dB}}}\right|^2 \tag{4.6}$$

The positive solution of β is shown in (4.7).

$$\beta = \cos(\omega_{3dB}) - 1 + \sqrt{\cos(\omega_{3dB})^2 - 4\cos(\omega_{3dB}) + 3} \tag{4.7}$$

From there, we can derive final solution as in Equation (4.8) where β is smoothing factor and ω_c is the cut-off frequency of the filter.

$$\beta = -y + \sqrt{y^2 + 2y}, \qquad y = 1 - \cos(\omega_c) \tag{4.8}$$

As it is mentioned before, the main strength of this filter is its simplicity. Equation 4.9 can be derived from Equation (4.3) and this equation is very suitable for parallelization.

$$y+ = \beta(x - y) \tag{4.9}$$

For this study, cut-off frequency ω_c is selected as 25 MHz resulting in $\beta = 0.0136$. α can be found using β from the relation $\alpha = 1 - \beta = 0.986$. According to these parameters, FFT outputs are smoothed and send to the RFNoC network whose next destination block is *Keep 1 in N*.

4.2.3.2. Keep 1 in N. Keep 1 in N block performs the down-conversion operation on the incoming digital signal. It keeps 1 sample from N incoming samples. Vectors can also be used instead of individual samples. This block was necessary because the sample rate in FPGA is 40 MS/s and CPU can handle up to 8 MS/s, which depends on the application. Therefore, the sample rate should be reduced before samples are sent to the CPU. If individual FFT samples are discarded, frequency information would be lost. This block helps to reduce the sample rate without losing any frequency information because it discards STFT vectors, not individual samples.

Figure 4.4 demonstrates how keep 1 in N block drops STFT vectors. This down-sampling procedure decreases the sample rate by N while preserving frequency bins for each STFT segment. N is selected as 18 by testing different values for CPU overflow. If lower values are used, overflow occurs in the CPU. Thus, the sample rate through CPU is $40MS/s/18 \approx 2.3MS/s$. Consistency of Frequency content is extremely important for accurate classification of 802.11b/g/n signals.

The next block is *head*, which stops the GNU Radio application, after acquiring M samples. This will be used to collect the desired sample count for autonomous data collection applications. Gathered samples are written to file for preparing data for the neural network.

4.3. Preprocessor

In this section, data pre-processing steps will be described. To extract features from the frequency domain, STFT vectors are given to CNN as input. However, if these vectors are fed to the network without any processing, computation will be very

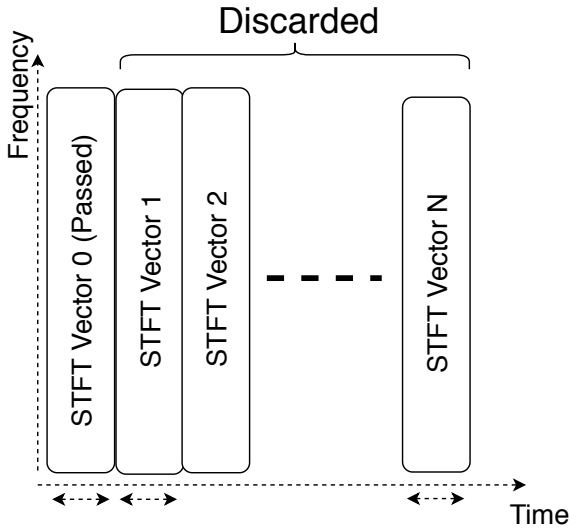


Figure 4.4. Keep 1 in N Block Drops Samples in Time Domain and Keeps All Information in Frequency Domain.

difficult since CPU receives data at a 2.3 MS/s rate.

As pre-processing steps, STFT vectors are used to generate spectrograms. After that, normalization is performed on spectrograms.

4.3.1. Spectrogram

Since they provide with good visualization of the spectrum, spectrograms [74] have been used in many classification tasks in literature such as classifying hand movements using EMG analysis [75], identification of smart jammers [25] and music classification [76] In this work, spectrograms will be employed to classify 802.11b/g/n signals.

The spectrogram is an image that represents the spectrum of a signal with respect to time [77]. It is computed by applying window functions such as Hann, Hamming, and Rectangular to time domain signals followed by performing STFT to time segments with a specific sample size. These segments, which STFT are employed, are combined by allowing some overlap between them.

STFT of a signal is taken by using equation (4.2) where L is window length, $\omega[m]$ is window function and $x[m]$ is input sample. In this implementation, the Rectangular window function was used because of hardware limitations. Inputs are grouped into segments of 512 samples. The frequency difference between consecutive FFT bins can be found using Equation (4.10) where κ, L and Δ_f are total frequency bandwidth, window length, and frequency interval, respectively. In our case, $\kappa = 40\ MHz$ and $L = 512$, then frequency resolution can be found as $\Delta_f = 78\ kHz$. This means, our implementation can detect OFDM subcarriers in 802.11g/n where subcarrier spacing is 312.5 kHz.

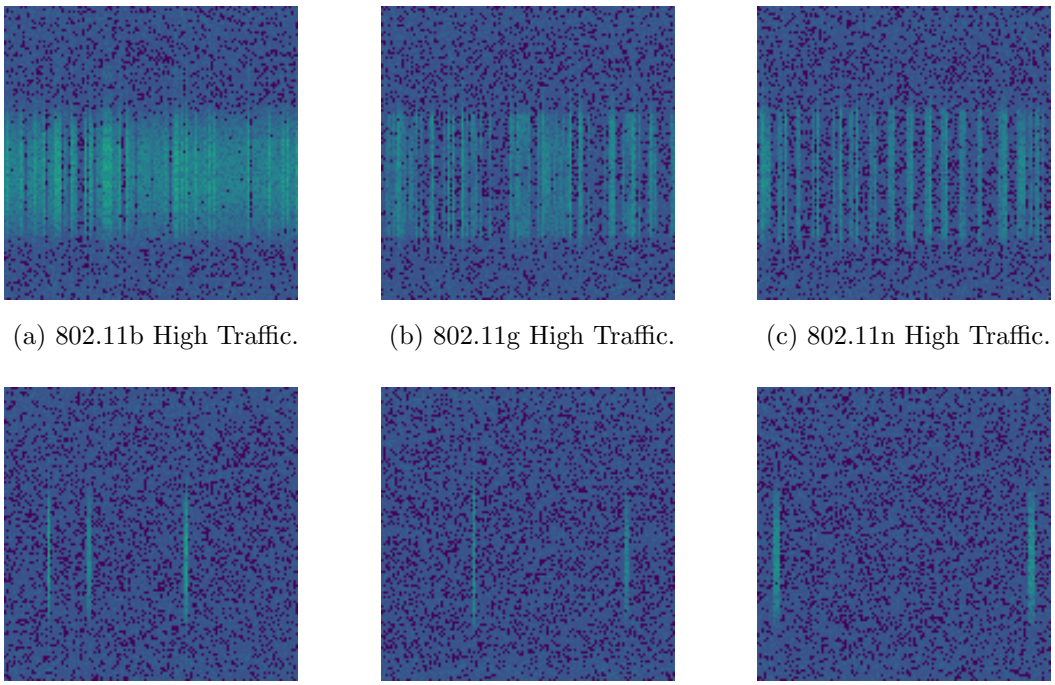
$$\kappa = L\Delta_f \tag{4.10}$$

Spectral energy density, which is found by taking a magnitude square of STFT, can be defined as spectrogram [78]. This is shown in equation (4.11).

$$\bar{P}(n,k) = \left|X(n,k)\right|^2 \tag{4.11}$$

Spectrograms are generated on CPU using the software, which collects 512 STFT vectors and combines them. Each STFT vector contains 512 samples. STFT vectors are obtained with Rectangular window function without any overlaps. Duration of each spectrogram image can be acquired using equation (4.12) where t_{STFT}, L, $t_{spectrogram}$, f_s and k are time duration of each STFT vector, window length, time duration of each spectrogram image, sampling frequency and count of STFT vectors which are used to generate spectrograms, respectively. Using the equations, time durations are found as $t_{STFT} = 12.8\mu s$ and $t_{spectrogram} = 6.6ms$. An important note is that the discarded STFT vector durations do not contribute to the time duration of the spectrogram image. Normally, 1 sample is kept from every 18 samples, thus the real duration of one spectrogram image is found as $18\, t_{spectrogram} = 118ms$.

$$t_{STFT} = \frac{L}{f_s}, \qquad t_{spectrogram} = k\, t_{STFT} \tag{4.12}$$



(a) 802.11b High Traffic. (b) 802.11g High Traffic. (c) 802.11n High Traffic.

(d) 802.11b Low Traffic. (e) 802.11g Low Traffic. (f) 802.11n Low Traffic.

Figure 4.5. Spectrogram Images for Active and Passive Channels.

Another reason why we use spectrograms is reducing data size. Normally, each spectrogram contains $512x512 = 262144$ samples, which was a size of $2\,MB$. After conversion to a spectrogram image whose dimensions are $128x128x3$, data size is reduced by 81%. This makes a significant improvement in efficiency and computation speed. In Figure 4.5, 802.11b/g/n spectrogram images are shown. Dataset is collected for high, which contains burst data and low traffic, which contains only protocol packets between modem and receiver. Images are drawn on a logarithmic scale. 6000 spectrograms are generated for each class which is b, g, and n. From these 6000 images, 3000 are high traffic and the remaining 3000 are low traffic data. These images are given to CNN as inputs after normalization.

4.3.2. Normalization

In classification algorithms scaling of data is very important for increasing convergence speed and accuracy. If data are not scaled well, some of the weights will affect the network in a specific direction. To avoid dependency on these weights, it is rec-

ommended to normalize (standardize) data [79]. Data normalization means mapping data to a small regular range such as [0,1] or [-1,1]. The normalization of data can speed up the training phase and increase accuracy in Neural Networks [80].

The two types of normalization that are mostly employed in machine learning tasks are *min-max normalization* and *z-score normalization*. In the former, data are scaled using minimum and maximum values that keep relation with original data. It is shown in equation (4.13) where y, x, min_A and max_A are normalized sample, input sample, minimum and maximum of data, respectively [80].

$$y = \frac{x - min_A}{max_A - min_A} \tag{4.13}$$

In our work, *z-score normalization* is employed, which is defined by equation (4.14) where y, x, μ_A, σ_A are output sample, input sample, mean and variance of data, respectively. Z-score normalization makes 0 mean and 1 standard deviation. In Figure 4.6, z-score normalization of spectrogram image is shown. This is the final version of data that will be given as input to CNN.

$$y = \frac{x - \mu_A}{\sigma_A} \tag{4.14}$$

4.4. Convolutional Neural Network Architecture

CNN is a very popular neural network type in the machine learning field mainly on images and videos, which contain more than one-dimensional data. First CNN is designed by Kunihiko Fukushima to achieve Visual Pattern Recognition task on hand-written digits [81]. Although many years passed since the first CNN was designed, CNN became more popular in recent years because of the enhancement of hardware, which provides much more computational capability.

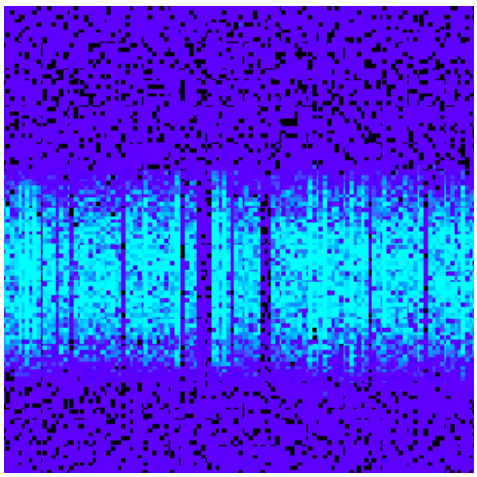


Figure 4.6. Z-Score Normalization of Spectrogram Image.

CNN consists of convolutional layers, which contain filters that are applied to inputs to extract local features. CNN can extract more complex features by using several convolutional layers consecutively. In [82], the authors showed how CNN sees the world by working on the ImageNet dataset. First layers capture features like corners, lines, edges, and last layers extract more complex features such as faces, legs, and objects. Convolution property of CNN makes it a reasonable choice for applications such as object detection [83], face recognition [84], image classification [85], sound [86], music and RF signal classification [87].

In this work, CNN is employed to classify 802.11b/g/n RF protocol signals. Normalized spectrogram images are fed to CNN as inputs for network training. CNNs generally contain several layers such as convolutional, max pooling, fully connected and in last years batch normalization [88].

Two important terms that represent the properties in the machine learning field are model parameters and model hyperparameters. Hyperparameters are the values, which the designer proposes before training to optimize the model such as the kernel size of a convolutional layer, the learning rate of the optimizer, etc. On the other hand, model parameters are the properties of the model. Model parameters such as weights of kernels and biases of neurons can be learned and updated by training.

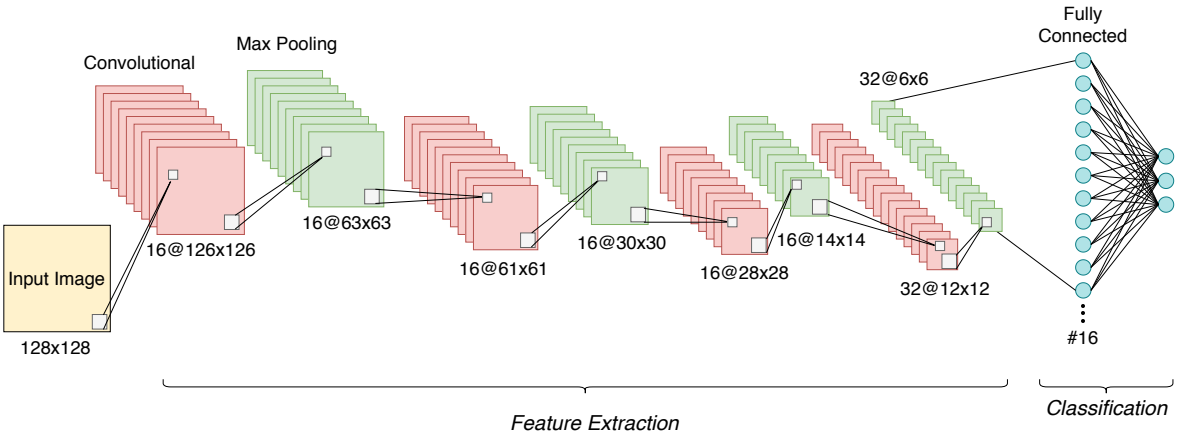


Figure 4.7. Designed CNN Architecture.

The designed CNN architecture is shown in Figure 4.7. It includes convolutional, max pooling, RELU activation, batch normalization, and fully connected layers. These layers are explained in the following subsections in more detail.

4.4.1. Convolutional Layer

The convolutional layer performs convolution operation on input by using a set of filters. In our work, the 2D convolution operation is employed, which is shown in equation (4.15) where Y, X, ω are output, input, and filter of convolution operation, respectively. The filter is called as *kernel* and output of convolution operation is in addition referred as *feature* or *activation map* [89].

$$Y(i,j) = (X * \omega)(i,j) = \sum_{m} \sum_{n} X(m,n)\omega(i-m, j-m) \tag{4.15}$$

The filter is a 2D array whose values are called kernel weights. Filter weights are often randomly initialized and they are updated by the backpropagation process while training the network. Each kernel extracts some features from images like edges, curves or more complex features like faces by updating weights.

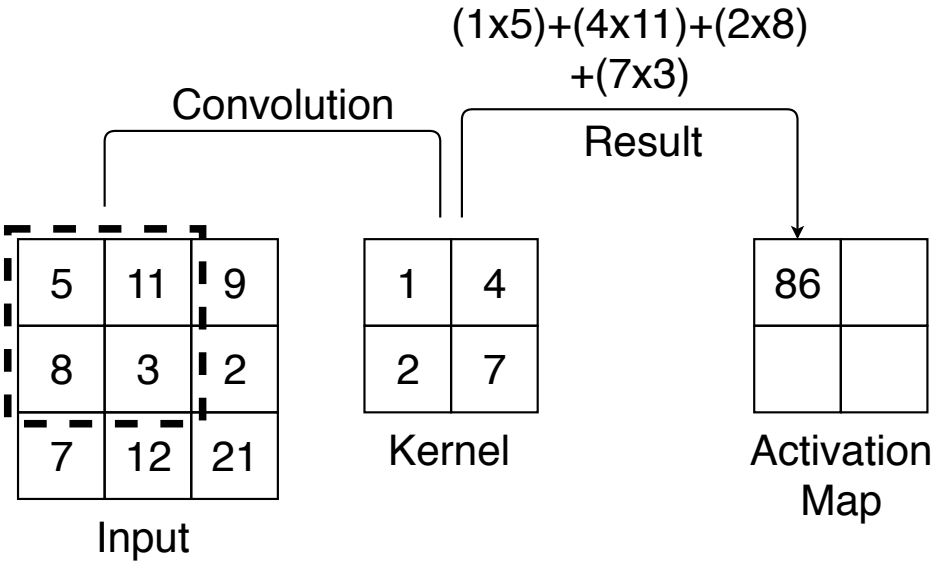


Figure 4.8. 2D Convolution Operation.

In a convolution operation, the kernel is slid across the width and height of input; it performs a dot product between the kernel and corresponding input values to generate one activation map value. This operation repeats until the entire image is scanned with the kernel. The acquired activation map presents the response of the input image to the corresponding kernel. For instance, if kernel represents an edge; when this kernel is convolved with an image that contains some edges, the activation map of this convolution operation will contain high values. Example of 2D convolution operation is shown in Figure 4.8.

The depth of the convolution kernel is always equal to the depth of input. The kernel is slid over input by specific step size which is called *stride* hyperparameter. Stride is generally taken as 1, but higher values can also be used to reduce computation load. Another important term in the convolutional layer is *padding* hyperparameter. Padding allows control over the size of the output activation map. If no padding is provided, the size of the activation map will be smaller than the input size. This means information on the borders will make less effect on activation map compared to information on the middle. Zero-padding improves performance by keeping information on the borders. Output size of activation map can be found using equation (4.16) where

$H'xW'$ is activation map size, HxW is input size, F is filter size and S is stride size. [90].

$$H' = \left\lfloor \frac{H - F + S}{S} \right\rfloor, \qquad W' = \left\lfloor \frac{W - F + S}{S} \right\rfloor \tag{4.16}$$

Hyperparameters are selected with experience or by using former proven results. However, smaller filter sizes are often used because of the decreasing number of learnable parameters and provide more patterns from local regions such as parts of objects in image [90].

4.4.2. Pooling Layer

The pooling operation performs a function that reduces the size of the input activation map. The function of the pooling layer can usually be average or maximum pooling. The pooling layer significantly reduces the data rate of input; this increases computational efficiency. This down-sampling operation yields making model invariant to small changes in the input. This helps the generalization of the model, which provides higher accuracy on the test dataset [89].

In this work, the max-pooling operation is used in pooling layers. This operation selects maximum value on the corresponding reception field of input; it also uses a nxn filter. *filter size* and *stride* are hyperparameters of this layer. Output size of this layer can be found using equation (4.17).

$$H' = \left\lfloor \frac{H - F}{S} \right\rfloor, \qquad W' = \left\lfloor \frac{W - F}{S} \right\rfloor \tag{4.17}$$

4.4.3. Activation

Activation layers are usually added to introduce non-linearity to network. Non-linearity is important for learning non-linear features. Designing a multi-layer network makes no sense without activation functions because it behaves like a single layer if all functions are linear. There are many activation functions such as Sigmoid, Tanh,

Rectifier Linear Unit (ReLU), Algebraic Sigmoid, etc. In this work, ReLU activation function is used in hidden layers due to its simplicity. Researchers showed that ReLU activation is much faster than others for large networks [91]. ReLU activation function is shown in equation (4.18).

$$f(x) = max(x, 0) \tag{4.18}$$

Also, Softmax activation is used in the output layer. When classifying the results, ReLU is not very useful. The softmax function provides a categorical probability distribution, which shows the true probability of each class. So, the network can predict the true value for the current forward-propagation part. It can be calculated using equation (4.19) where h_j is an input of the output layer and K is the number of classes.

$$\sigma(h)_j = \frac{e^{h_j}}{\sum_{k=1}^{K} e^{h_k}} \tag{4.19}$$

4.4.4. Fully Connected (FC) Layer

Feature extraction in CNN is performed by convolutional layers. After convolutional layers, the output is flattened and passed as an input to FC layers. FC layers learn non-linear combinations of these features. It basically checks which high-level features correlate most with the corresponding class. Basic FC Layer architecture is shown in Figure 4.9.

FC layers contain neurons that have connections to all of its inputs and outputs. These connections are called weights. Basically, the output of the neuron is calculated using inputs, weights, and bias. Bias can sometimes be considered as an additional weight with a constant input value of 1 and it is used to shift. The output of the neuron is passed from a non-linear activation function to introduce non-linearity to the network. The output of a neuron can be calculated using Equation (4.20) where ω_i is

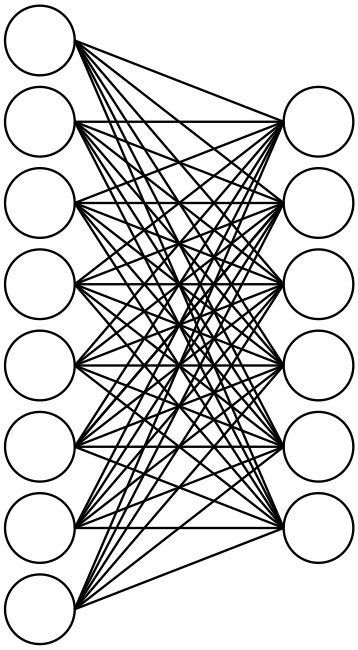


Figure 4.9. Basic Fully Connected Layer.

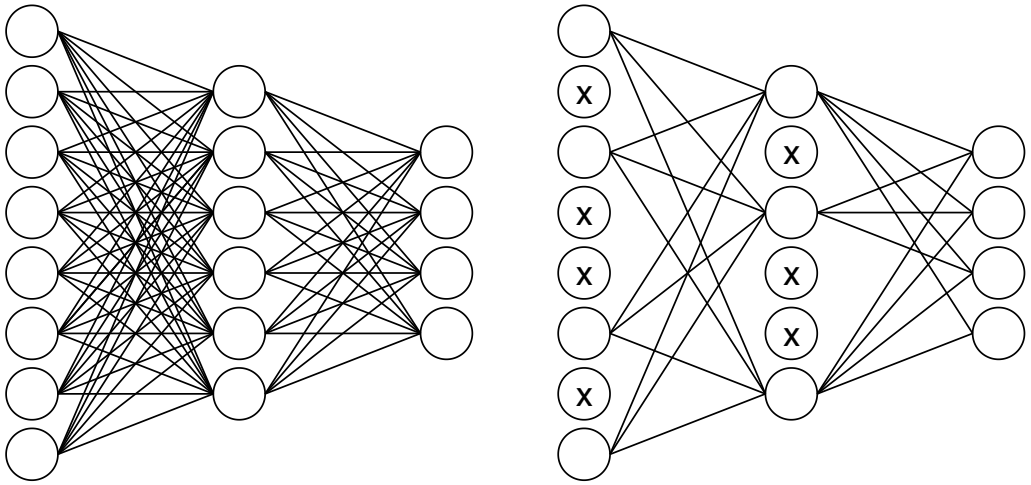
weight vector, x_i is input vector, b is bias of neuron, z is output of neuron and f is non-linear activation function. In this work, two FC layers are used. In the output FC layer, the softmax activation function is used to predict results; ReLU activation function is used in the hidden layers. These activation functions are explained in section 4.4.3.

$$z = f(\sum_i \omega_i x_i + b) \tag{4.20}$$

Here, ω and b are learnable parameters that are updated during backpropagation. In the next sections, the update process of these parameters will be explained.

4.5. Regularization

Neural networks have many learnable parameters that help to find a pattern from the input dataset. However, the network may learn more than necessary. This situation is called over-fitting. Over-fitting means that the network model performs better on the training dataset, but performs poor on the test dataset. It can be said that the network is memorizing its inputs. Regularization techniques are developed



(a) NN without Dropout. (b) NN with Dropout.

Figure 4.10. Dropout Layer Visualization.

to prevent over-fitting. Regularization means that making a slight modification on the model for increasing the training error and decreasing the generalization error. Better generalization increases the success rate of the model on unseen data. The most commonly used regularization methods are Dropout, L1 and L2 Regularization, Data Augmentation and Batch Normalization.

In our work, Dropout and Batch Normalization regularization techniques are employed to prevent over-fitting. In the following sections, these methods will be explained.

4.5.1. Dropout

One of the most popular methods for neural network regularization is Dropout. Dropout drops neurons randomly in FC layers. Dropping means that the contribution of dropped nodes to the output of the neuron is decreased to zero [92]. In Figure 4.10, visualization of Dropout is shown. When dropout is not used, the network will adapt to the learned features of specific neurons. This will prevent generalization, which means failing on the prediction of test data. Dropout should not be used on prediction since it causes noise [93].

4.5.2. Batch Normalization

In machine learning, normalization methods are applied to inputs for bringing data to a common scale. Ioffe *et al.* [88] proposed a method for normalizing hidden layer outputs to reduce internal covariance shift, which means changing the distribution of activations in hidden layers. This distribution change reduces convergence speed due to the fact that the network tries adapting to new distributions. It is desired to reduce internal covariance shift in machine learning. When the inputs of layers are whitened (zero mean, unit variance), which means eliminating the internal covariance shift, the network converges faster [94].

Batch normalization is a significant process that is used in state-of-the-art CNN designs. It tries normalizing hidden layer activations in each iteration to minimize internal covariance shift; it also stabilize the network. It is beneficial for several reasons. The network training process becomes less responsive to hyperparameter changes. It increases the convergence speed of the network and contributes to the regularization of the network.

Batch normalization can be defined using the equations below. In (4.21) and (4.22), mean and variance of m input images are calculated. Here, μ_{x_i} and ${\sigma^2}_{x_i}$ represent batch mean and variance, respectively. Here, κ is values of x over mini-batch $x_{1...m}$.

$$\mu_\kappa = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{4.21}$$

$${\sigma^2}_\kappa = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\kappa)^2 \tag{4.22}$$

In (4.23), $\hat{x_i}$ is normalized activation value and in (4.24), y_i is its linear transformation. Here, normalization itself is not enough, since it can change what the layer represents. Thus, normalized values are scaled and shifted to learn parameters of γ and β. These

parameters are updated during backpropagation. By learning these parameters, the mean and variance of layer input activations can be converted into any value the network desires.

$$\hat{x_i} = \frac{x_i - \mu_\kappa}{\sqrt{{\sigma^2}_\kappa + \epsilon}} \tag{4.23}$$

$$y_i = \gamma \hat{x_i} + \beta \tag{4.24}$$

4.6. Optimization

Until this section, all the layers in the CNN developed for this work were covered. There are learnable parameters in some layers such as weights of connections, biases, batch normalization parameters, etc. These parameters should be set to appropriate value to complete the training process.

Supervised Learning which has the training dataset of the network labeled before starting training is the learning type of the network in this work. Labeling means identifying input and telling the network what that input is. There is also another learning type that is called Unsupervised Learning. In this type, inputs are not labeled and the network learns itself what they are and groups the input dataset into clusters.

In general, the dataset is divided into three segments as training, test, and validation. The test data, which are hidden from the network, is used when the training is finished. Using test data helps to observe how good the model can predict the results of unseen data. The training data is used during training the model to update weights and the validation data is used by the designer for fine-tuning of hyperparameters. During the training of the network, the trained model is tested using the validation dataset in each epoch. The epoch means passing all samples one time through the network. The designer can follow the training behaviour of the network using a validation dataset and can interrupt the training to change hyperparameters when over-fitting or

under-fitting occurs. As a rule-of-thumb, if you have enough data for training, 80% of data is used for the training and 20% for the test. The validation dataset is 20% of the training dataset.

Another data splitting method for the training data is *K-Fold Cross Validation*. In this method, the data is split into K parts and one of them is used for the validation and others for the training. This is performed for K different sets. This method provides testing the trained model on different test sets and leads to a more stable estimation [95].

In this section, the optimization part of the CNN training will be explained. When an input is given to the neural network and passes from hidden layers and generates output on the last layer, this process is called *forward-propagation*. This process does not update the weights; therefore, no training occurs. To train the network, a loss should be calculated for predictions of the network for each input sample. Then, this loss should be minimized by optimization algorithms. When the loss minimization is sufficient, training of the network will be completed.

Optimization algorithms calculate gradients to update the weights of the network. Gradients are calculated using *backpropagation* method that applies the chain rule to compute the derivative of the loss function with respect to each weight. Optimization algorithms update weights using gradients and the parameter that is called *learning rate* (η).

4.6.1. Loss Function

As it was mentioned in the previous section, optimization algorithms use loss function to measure how well the network makes predictions on labeled training data. The loss function is optimized by updating the network weights so that the difference between the actual and the predicted outputs are minimized. There are many loss functions such as Mean Square Error, Mean Absolute Error, Multi-Class SVM Loss, Cross-Entropy Loss.

In our network, the Categorical Cross-Entropy loss function is used to estimate the quality of predictions. Cross-Entropy Error for binary classification can be calculated using (4.25) where y is the correct output and p is the probability of each category.

$$L(p, y) = -[y \log(p) + (1 - y) \log(1 - p)] \tag{4.25}$$

The Cross-Entropy loss function can be generalized for multi-class classification as in equation (4.26) where N total number of neurons in output layer. The purpose of the optimization algorithm is minimizing this loss function.

$$L(p, y) = -\sum_{k=1}^{n} y_n \log(p_n), \qquad n \in [1, N] \tag{4.26}$$

4.6.2. Backpropagation

Other important parts of the neural network training are backpropagation and weight update. It is mentioned in the previous sections that the first layers of CNN learn to check for edges, curves, etc. Actually, weights of filters are updated by optimization algorithms using the backpropagation method for generating filters that detect edges or curves.

Backpropagation is a method for calculating the gradient of the loss function with respect to the different weights. Backpropagation calculates gradients using the chain rule of differential calculus. The calculated gradients are used to update weights. Here, an important distinction is that backpropagation is a method for calculating gradients, it does not update the weights of the network directly. The weight update process is performed by optimization algorithms such as gradient descent, Adaptive Moment Estimation (Adam), Adaptive Gradient Algorithm (AdaGrad), etc. [96].

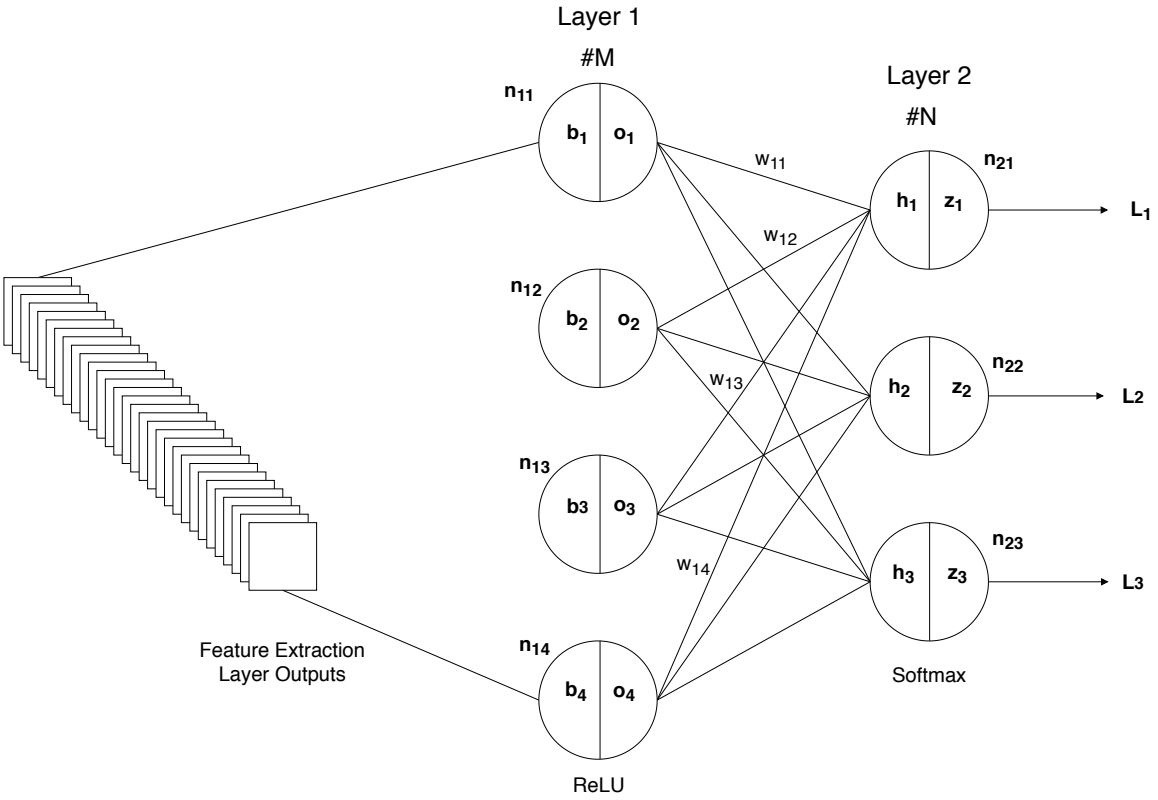


Figure 4.11. Backpropagation Method.

In Figure 4.11, visualization of backpropagation method is shown. For better illustration, the last two layers of designed CNN are shown for backpropagation. This structure is the same as our network but a simplified version. Activations are ReLU in layer 1 and softmax in layer 2. Calculation of backpropagation for weights of neuron n_{21}, which are $w_{[11\,\ldots\,14]}$, will be performed for better understanding. To find gradients for weights, partial derivative of loss function L with respect to weights w_{jt} will be calculated using equation (4.27) where z_j, h_j, w_{jt}, y_j, L are activations of neuron j in layer 2, output of neuron j in layer 2, weight between layer 1 and 2, correct output for neuron j and loss of the network, respectively.

$$\frac{\partial L(z_j, y_j)}{\partial w_{jt}} = \frac{\partial L(z_j, y_j)}{\partial z_j} \frac{\partial z_j}{\partial h_j} \frac{\partial h_j}{\partial w_{jt}} \tag{4.27}$$

In equation (4.27), chain rule for calculation of gradient for weight w_{jt} is shown. The backpropagation algorithm calculates all elements in the chain rule and find the final

result. First, the partial derivative of loss $L(z_j, y_j)$ with respect to activation output z_j will be found.

$$L(z_j, y_j) = -\sum_{j}^{N} y_j \log(z_j) \tag{4.28}$$

$$\frac{\partial L(z_j, y_j)}{\partial z_j} = -\sum_{j}^{N} \frac{y_j}{z_j} \tag{4.29}$$

In (4.28), multi-class cross-entropy loss function of the network is calculated and in (4.29), the partial derivative of the loss function with respect to activation z_j is calculated.

In (4.30), Softmax activation result of the output layer is calculated where z_j is neuron that softmax is calculated for and N is total neuron count that is also called class count in the last layer. Here, there are two solutions of derivative of Softmax activation output z_j with respect to the neuron output h_j, which are shown in (4.31). Now, two elements of the chain of derivatives are found. The partial derivative of the loss function with respect to neuron output can be calculated. This calculation is shown in (4.32).

$$z_j = \frac{e^{h_j}}{\sum_k^N e^{h_k}} \tag{4.30}$$

$$\frac{\partial z_j}{\partial h_j} = \begin{cases} z_j(1 - z_j), & j = k \\ -z_j z_k, & j \neq k \end{cases} \tag{4.31}$$

$$\begin{aligned}\frac{\partial L(z_j, y_j)}{\partial h_j} &= -\sum_j \frac{y_j}{z_j}\frac{\partial z_j}{\partial h_j} \\ &= -y_j(1-z_j) - \sum_{k\neq j}\frac{y_j}{z_j}(-z_j z_k) \\ &= -y_j + y_j z_j + \sum_{k\neq j} y_j z_j \\ &= z_j - y_j\end{aligned} \tag{4.32}$$

Finally, the partial derivative of the neuron output h_j with respect to the weight w_{jt} is necessary to complete the entire chain. Neuron outputs can be calculated using equation (4.33) where o_t is activation of neuron t in layer 1, w_{jt} is weight from layer 1 to 2 and b_j is the bias term of neuron j in layer 2 and M is the total neuron count in layer 1. Using this result, last element of the chain is calculated in equation (4.34).

$$h_j = \sum_t^M o_t w_{jt} + b_j \tag{4.33}$$

$$\frac{\partial h_j}{\partial w_{jt}} = \sum_t^M o_t \tag{4.34}$$

All required elements of the chain are calculated in the previous equations. The entire chain is calculated in (4.35).

$$\frac{\partial L(z_j, y_j)}{\partial w_{jt}} = (z_j - y_j)\sum_t^M o_t \tag{4.35}$$

In this section, the backpropagation process for weights between the last two layers was performed. The task of the backpropagation algorithm is calculating gradients and results are used by optimization algorithms to minimize the loss function.

4.6.3. Adam Optimizer

Optimization algorithms are employed to minimize the loss function of the neural network $L(z_j, y_j)$. Optimization algorithms calculate gradients of the loss function with respect to weights; they update the weights according to the gradient. There are many optimization algorithms such as gradient descent, AdaGrad [97], AdaDelta [98], Adam [99], RMSprop. In this work, Adam optimization algorithm is employed to minimize the loss function.

The Adam method computes individual adaptive learning rates for different parameters using the first and the second moments of the gradients. This method combines the advantages of Adagrad and RMSprop optimization algorithms. Adam uses the first moment (mean) and the second moment (the uncentered variance) of gradients to update the parameters. The first and the second moments can be calculated using equations (4.36) and (4.37), where m_t, v_t, g_t, β_1 and β_2 are mean, uncentered variance, gradient of current mini-batch and decay hyperparameters, respectively [100]. Decay hyperparameters β_1 and β_2 are assigned 0.9 and 0.999 in the original paper [99].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.36}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.37}$$

When m_{t-1} and v_{t-1} are initialized as 0's, it is observed that moment estimates are biased towards zero, especially in the first steps, and also especially when decay hyperparameter is close to 1. Thus, bias-corrected moment values $\hat{m}_t$ and $\hat{v}_t$ are generated by using equation (4.38) and (4.39) to counteract initial bias.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.38}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{4.39}$$

Final update process is shown in (4.40). Here, θ_{t+1} and θ_t are next and current values of weight. The learning rate η is also taken as 0.001 in the original paper [99]. This update method is also used in Adadelta and RMSprop. The authors show that Adam algorithm runs superior compared to other algorithms [100].

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon}\hat{m_t} \tag{4.40}$$

5. EXPERIMENTS AND RESULTS

Our system contains two steps that are the training and the real-time tests. In the training, the model is trained using spectrogram samples on a high-performance computer in the cloud and the trained model is tested on other unseen spectrogram samples to find the success rate of the model. In the real-time tests, the device is tested with real wireless modem signals for different data rates using a script that runs on the device.

Three different types of spectrogram images are acquired from raw IQ data as *8-bit RGB*, *8-bit grayscale*, and *16-bit grayscale*. These image datasets are used as inputs to the CNN and three different models are generated. All the models are employed in real-time tests on USRP. Training and test processes are explained in the following sections.

Table 5.1. Test Setup Features.

	Receiver (Rx)	**Transmitter (Tx)**
Device	Ettus USRP E310	Huawei HG531s V1 Modem
Analog Gain	0 dB	+20 dB
Distance	2 m	
Center Frequency	2.44 GHz	2.437 GHz (WiFi Ch. 6)
Analog Bandwidth	56 MHz	-
Sampling Rate	40 MHz	-

5.1. Training of Classifier Model

The CNN design, explained in section 4.4, is employed for the signal classification task. The designed model should be trained offline using generated spectrogram samples before putting the model into USRP E310 for real-time tests. The data col-

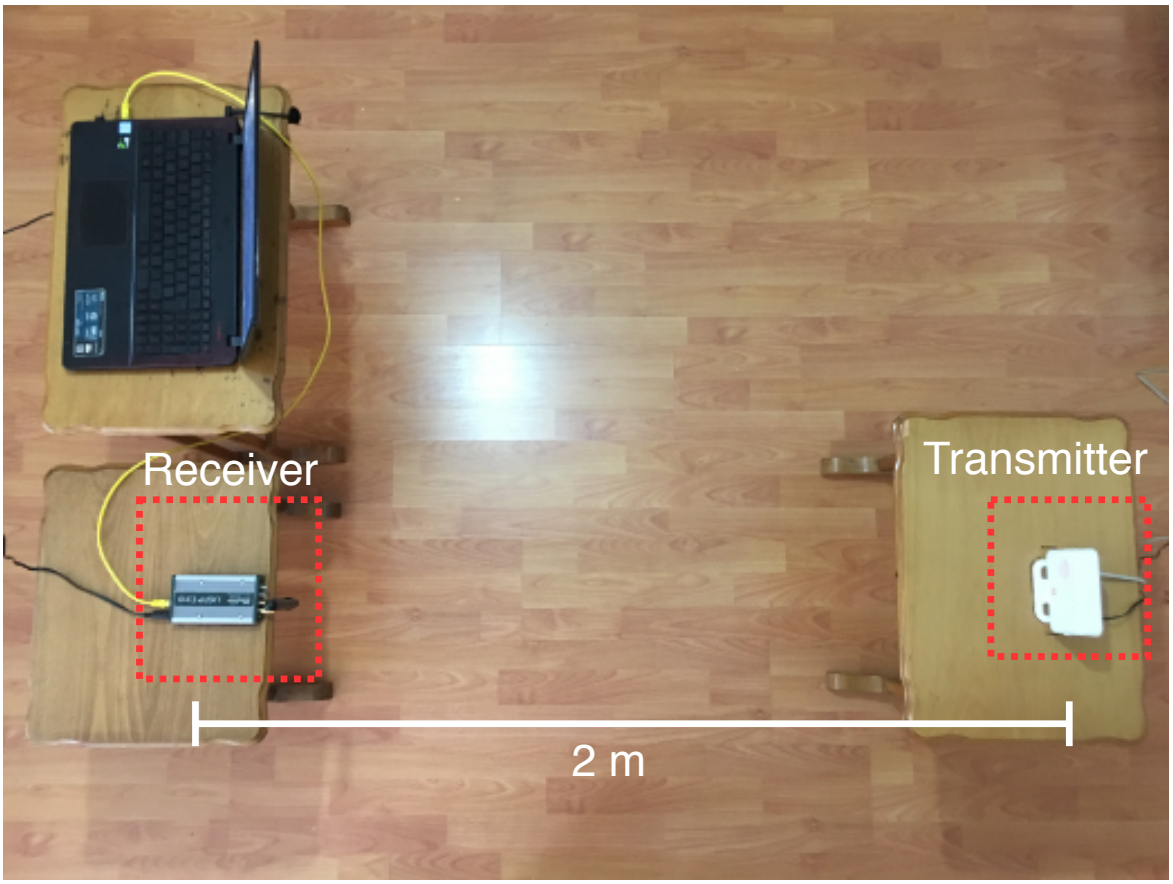


Figure 5.1. Test-Bed Setup.

lection task is important due to the importance of data in deep learning. Even if the designed neural network is capable, the result will not be a successful if the dataset is inadequate. Therefore, first, a test setup must be built for dataset collection. In Figure 5.1, the constructed test setup is shown. Here, USRP E310 is used as the receiver. An ADSL modem is used as 802.11b/g/n signal transmitter. Test setup features are shown in Table 5.1.

As stated before, an IIR filter is used in our application to smooth images. Single-pole IIR filter behaves as moving average filter; moving average filter is better in the frequency domain for reducing random noise [101]. First of all, three different datasets were collected to see the effect of passing the signal through a single-pole IIR filter on classification accuracy. In the first case, time domain signals were filtered before conversion to the frequency domain. In the second case, frequency domain signals were

smoothed by passing them through the IIR filter. In the last case, no IIR filter was applied. Table 5.2 shows that the highest test accuracy is obtained when frequency domain signals are smoothed by IIR filter and the worst results are obtained when time domain signals are filtered. This may be due to the fact that filtering loses valuable information in the time domain signal while smoothing emphasize the more relevant lower frequency contents.

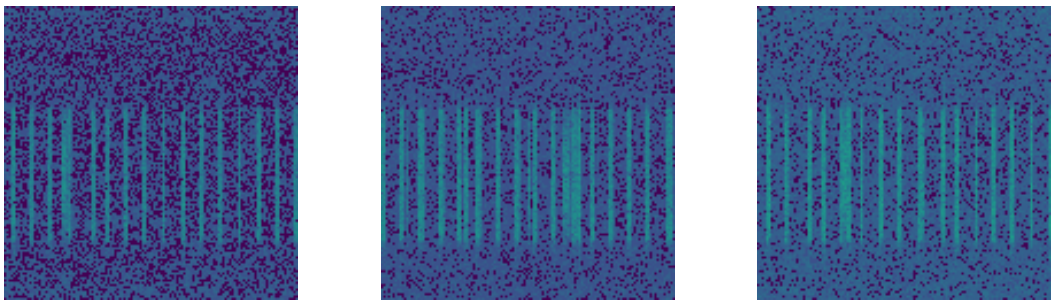
Table 5.2. Test Accuracies for different IIR filter Cases.

Time-Domain	**Frequency-Domain**	**No IIR**
80.2%	91.7%	85.4%

Spectrogram images for three cases are given in Figure 5.2. Here, it can be observed that the noise level is reduced when the IIR filter is used. In Figures 5.3, 5.4 and 5.5, confusion matrices for each cases are shown. The confusion matrix is a representation of the model's success on test data. In the ideal case, all predicted and true values should be equal for each class. In Figures 5.3, 5.4 and 5.5, it can be observed that some of the images are predicted false. The total accuracy of the model is calculated using these values in the confusion matrix. According to accuracy results, the best configuration for the IIR filter is in frequency-domain. It achieves an accuracy of 91.7% on the test set. As a result, all the classification tasks will be realized using the IIR filter in frequency-domain.

5.1.1. Dataset Collection

Dataset collection is an important part of deep learning tasks. Dataset collection is divided into two steps. In the first step, the dataset is collected for high traffic, which is generated by downloading a file from the internet and PC is connected to the modem. The modem will send WLAN packets via wireless communication that is configurable as 802.11b/g/n. Data is collected for the three classes as 802.11b, 802.11g and 802.11n.



(a) Time-Domain. (b) Frequency-Domain. (c) No IIR.

Figure 5.2. Spectrogram Images for Time-Domain IIR, Frequency-Domain IIR and without IIR.

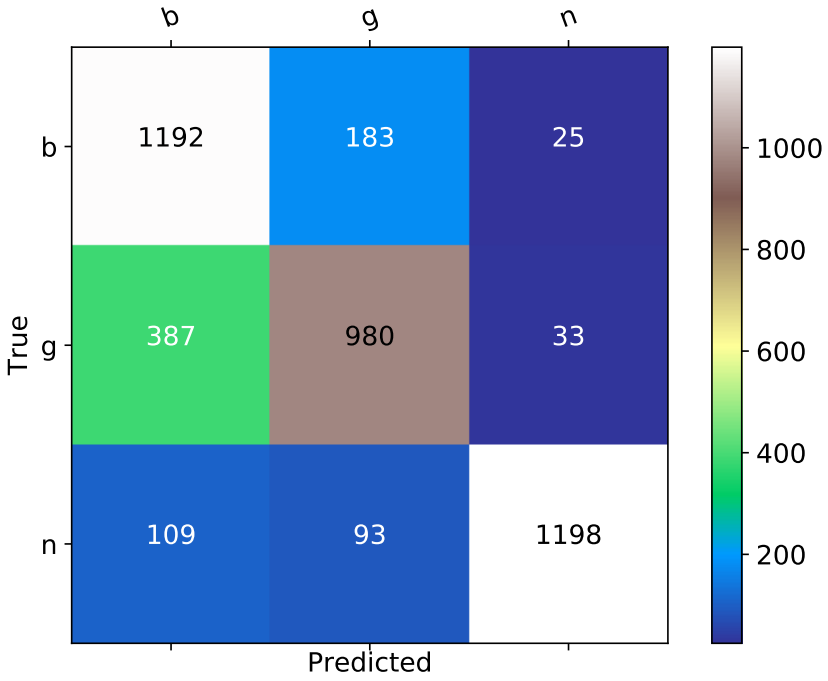


Figure 5.3. Confusion Matrix for Time-Domain.

First, a small dataset is collected for deciding the IIR filter configuration. After deciding the configuration, 22200 spectrogram images are generated for each class as 8-bit RGB, 8-bit grayscale and 16-bit grayscale for final training. The grayscale images only contain luminance information. RGB images include red, blue and green information. The spectrogram samples for high traffic RGB dataset are shown in Figure 5.6.

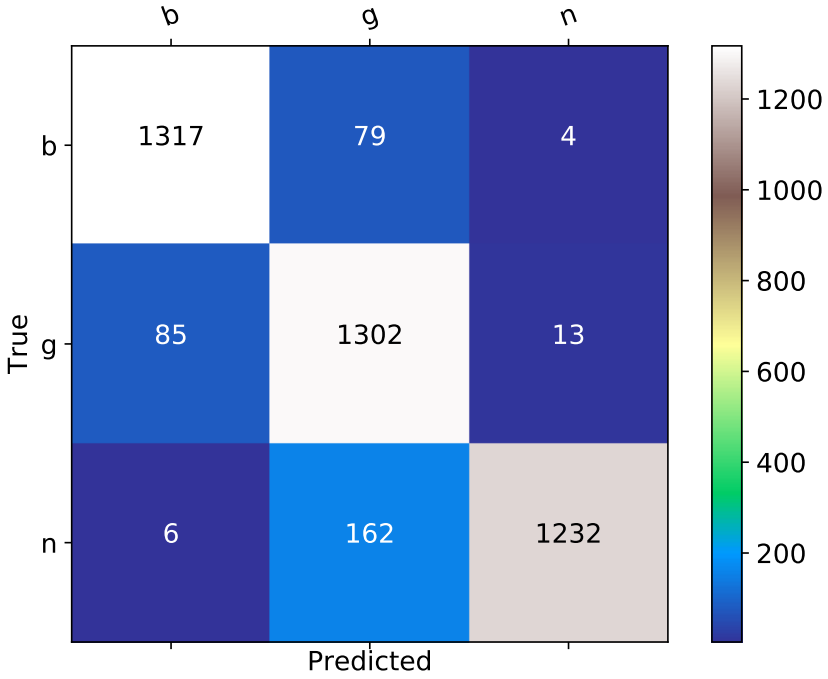


Figure 5.4. Confusion Matrix for Frequency-Domain.

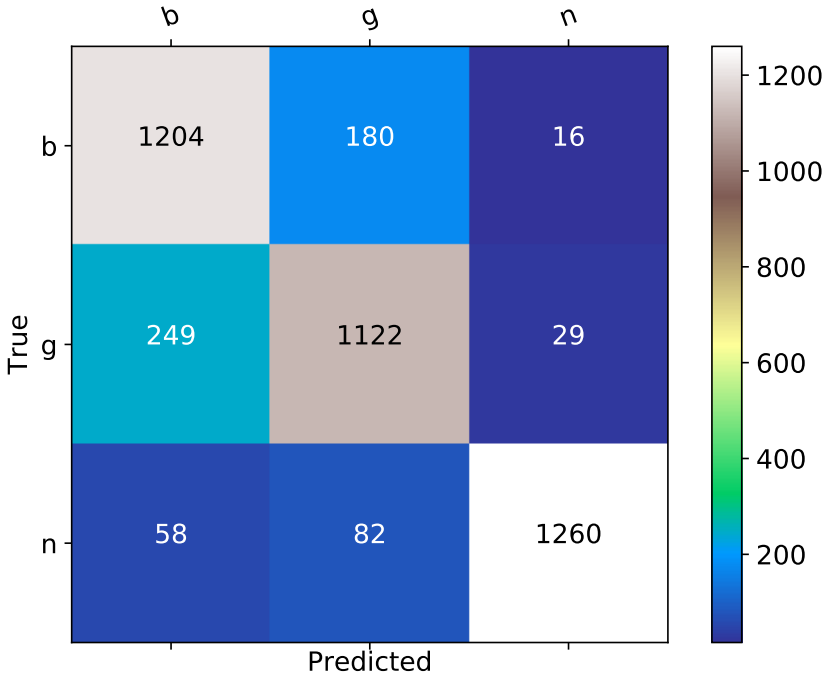
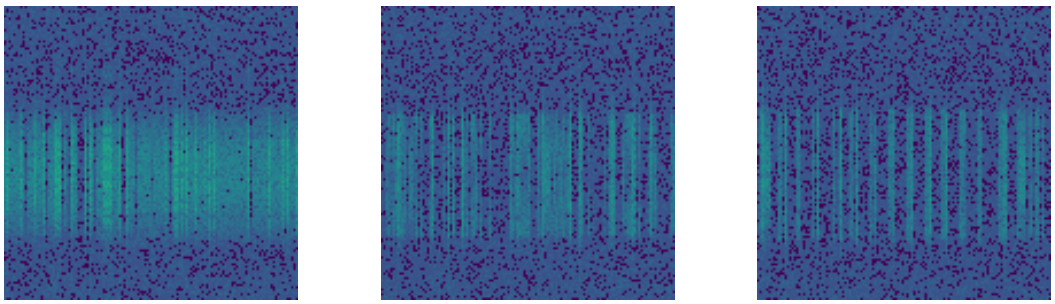
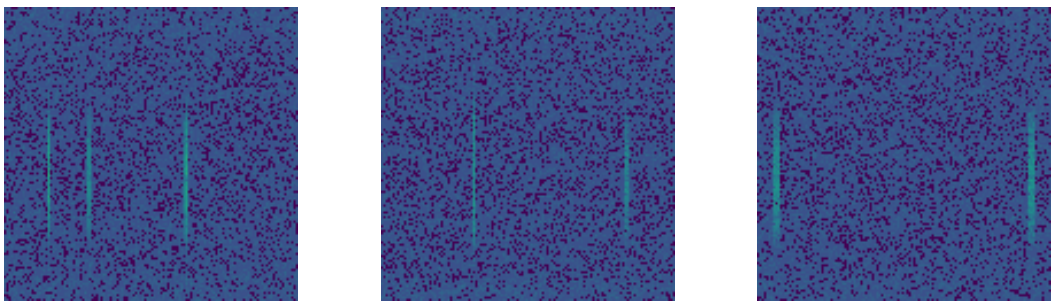


Figure 5.5. Confusion Matrix for No IIR.



(a) 802.11b. (b) 802.11g. (c) 802.11n.

Figure 5.6. Spectrogram Images for High Traffic Dataset.

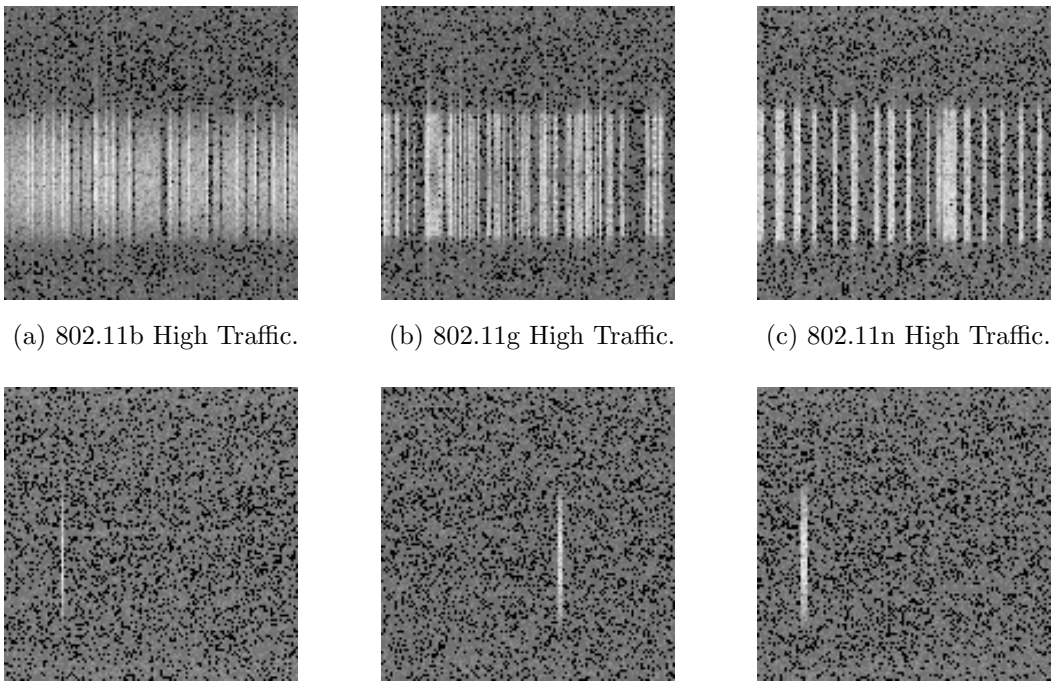


(a) 802.11b. (b) 802.11g. (c) 802.11n.

Figure 5.7. Spectrogram Images for Low Traffic Dataset.

In the second case, the dataset is collected for low traffic mode. In this mode, no file is downloaded from the internet and only the protocol packets appear in the spectrum. This results in very few packets in the spectrum. This type of dataset is intended to improve the adaptivity of the CNN model to different data rates. The final model can predict the signal type in high and low traffic environments. Sample spectrogram images for this dataset is shown in Figure 5.7, which reveals that much fewer transmissions occur in one spectrogram image.

A dataset is also prepared with grayscale images for high and low traffic samples. Grayscale images are coded with 8-bit and 16-bit; RGB is coded with 8-bit. Storing an image in grayscale results in a 33% reduction in image size for 16-bit and 66% for 8-bit, which also affects the computational load on the neural network. Sample spectrogram images for 16-bit grayscale dataset is shown in Figure 5.8. All images are normalized before feeding as input to CNN. Normalized grayscale and RGB images are shown in



(a) 802.11b High Traffic. (b) 802.11g High Traffic. (c) 802.11n High Traffic.

(d) 802.11b Low Traffic. (e) 802.11g Low Traffic. (f) 802.11n Low Traffic.

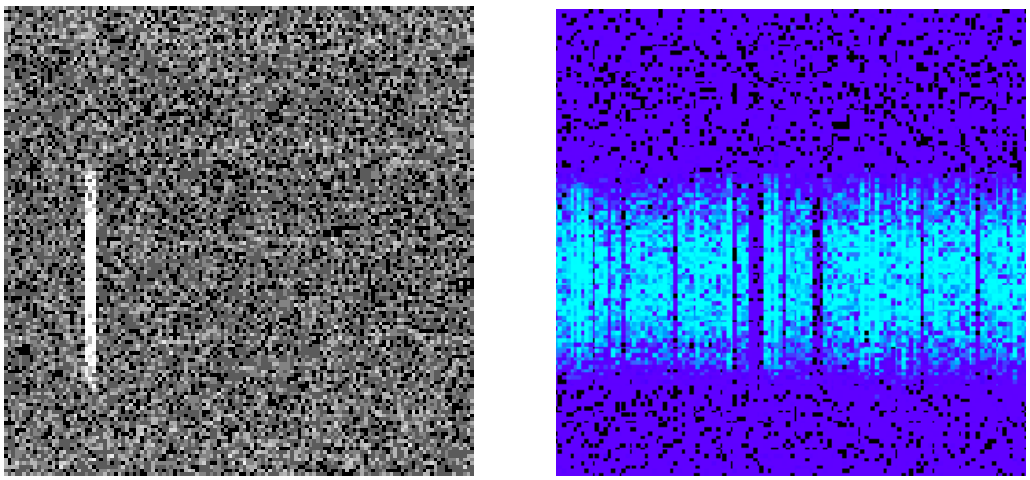
Figure 5.8. Grayscale Spectrogram Images for High and Low Traffic Dataset.

Figure 5.9.

After completing the dataset collection task, the dataset is separated into training, validation and test sets. The training set is used by the backpropagation algorithm to update weights for the training model. The validation set is employed to monitor the training process for hyper-parameter tuning. On every epoch (processing of all samples for one time), the current model is used to predict validation samples. Prediction results are used for monitoring over-fitting and under-fitting problems. The test set is used for examining the final model, which is generated after completing the training process. Distribution of dataset is shown in Table 5.3.

Table 5.3. Training Dataset Distribution.

Training	**Validation**	**Test**
65%	16%	19%



(a) Grayscale Image. (b) RGB Image.

Figure 5.9. Normalized Spectrogram Images.

5.1.2. Training Results for RGB Image Dataset

After completing dataset collection and separation of the dataset into training, validation, and test; training of CNN begins. CNN is trained in a high-performance computer in the cloud, which has Tesla V100-SXM2 GPU with 32 GB memory. Completing each epoch takes 30 seconds on this computer.

$$Accuracy = \frac{\#\ of\ Correct\ Predictions}{\#\ of\ Total\ Predictions} \tag{5.1}$$

Accuracy vs Epochs, Loss vs Epochs and confusion matrices are given as performance metrics. In Figure 5.10, Accuracy vs Epoch graph is given. Here, *acc* and *val_acc* are training and validation accuracies, respectively. Accuracy metric is calculated using equation (5.1). This model is trained for 100 epochs and its validation accuracy reaches to 92.2%. The figure also shows that there is no under-fitting or over-fitting in the model. The accuracy of the validation shows the same trend with the training. An example of the over-fitting situation is shown in Figure 5.11. Although training loss keeps decreasing, validation loss begins to increase after a certain point. This means that the model is over-fitted. The best way to overcome over-fitting is by collecting more data. If this is not possible, regularization techniques such as dropout, L1 and

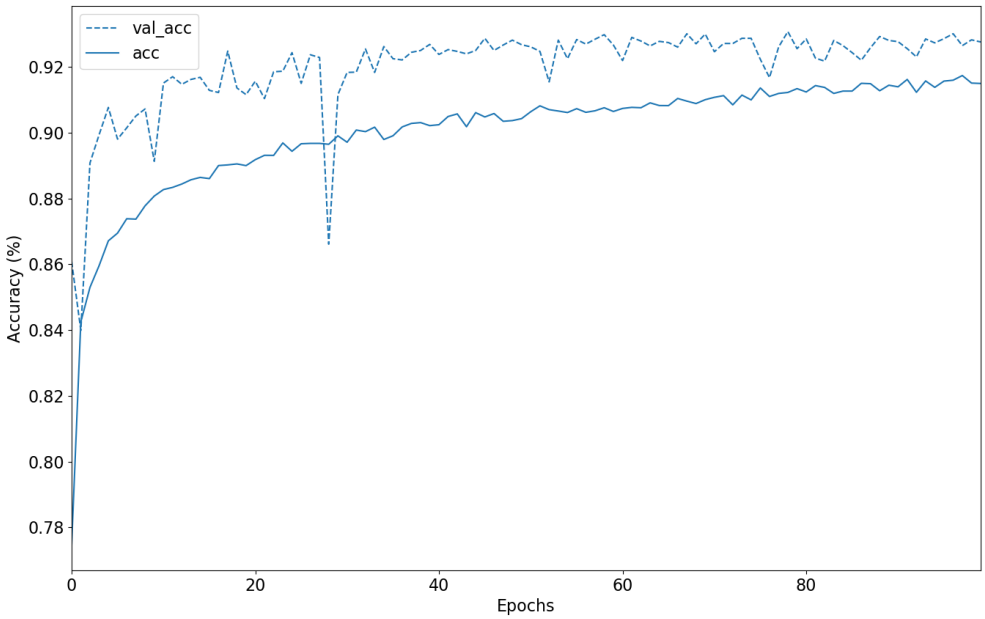


Figure 5.10. Accuracy vs Epochs for RGB Image Dataset.

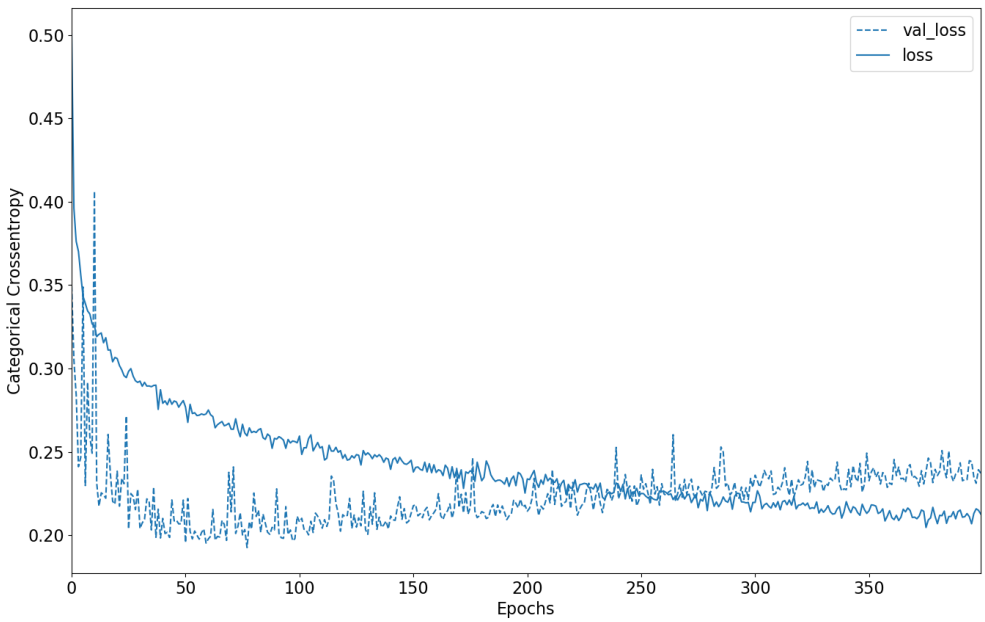


Figure 5.11. Over-fitting Model Example.

L2 regularization can be applied. Another method is early stopping, which refers to terminating the training process after a decided point to prevent over-fitting. In the figure, the model over-fits after approximately 60 epochs. Early stopping can be performed by stopping the network at the 60th epoch. Also, when there is no progress on validation accuracy for n epochs, training can be stopped. This is also another early stopping method.

The categorical Cross-Entropy Loss function is employed in the output layer of the CNN. Loss function with respect to epochs is shown in Figure 5.12 for RGB image dataset classification. The same trend in the accuracies can be observed here. Validation loss decreases with the epochs. Confusion matrix of RGB image dataset classification is shown in Figure 5.13. The overall accuracy of the test set for the RGB dataset is 90.0%.

5.1.3. Training Results for Grayscale Image Dataset

In the second case, images are converted to grayscale, which has data size and computation cost advantages. For 16-bit grayscale dataset, image size decreases by 33% and each epoch of training reduce to 25 seconds, which is 16% less than the time each epoch takes for RGB classification. In Figure 5.14, confusion matrix for 16-bit grayscale image training is shown. The overall accuracy of the model in the test dataset is 91.4%. In Figure 5.15 and Figure 5.16, accuracy and loss values with respect to epochs are presented, respectively. Over-fitting and under-fitting were not observed in training as it is seen from the figures. The trained model has achieved 91.5% accuracy in the validation set. 8-bit grayscale model also achieves 90.2% accuracy on test set.

Finally, classification results comparison of RGB, 8-bit and 16-bit grayscale image dataset is shown in Table 5.4. Grayscale classification accuracy is slightly better compared to RGB because the time-frequency information of the signal can only be presented with luminance values. RGB brings extra information that increases complexity. This also causes longer training time and larger data size. Grayscale model with 16-bit resolution is also better than 8-bit resolution.

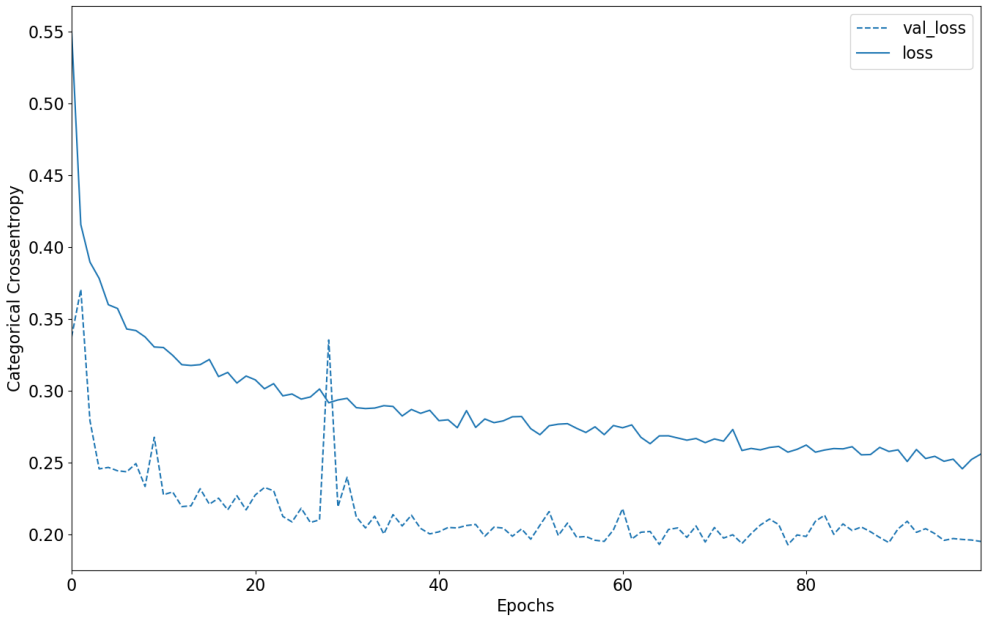


Figure 5.12. Loss vs Epochs for RGB Image Dataset.

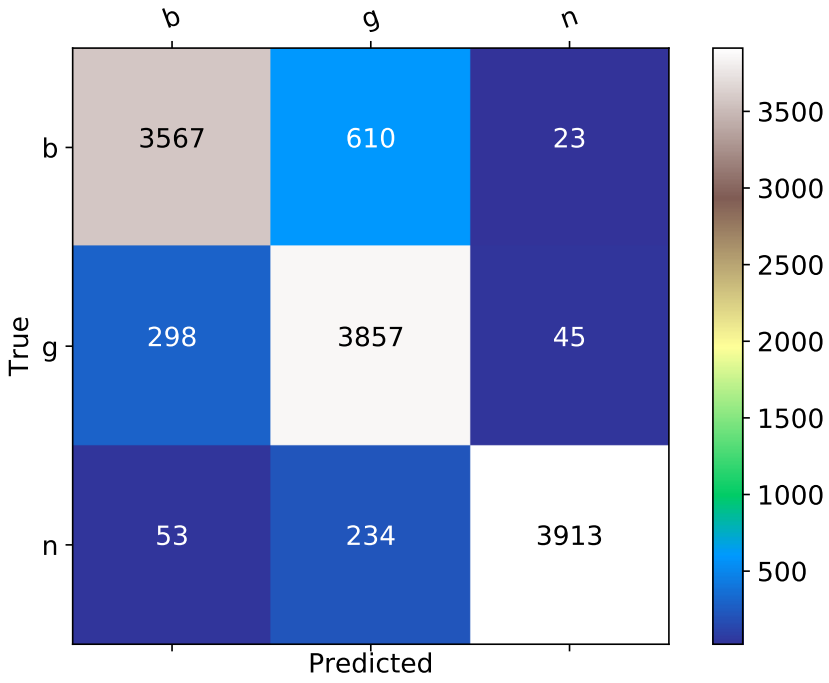


Figure 5.13. Confusion Matrix of RGB Image Dataset.

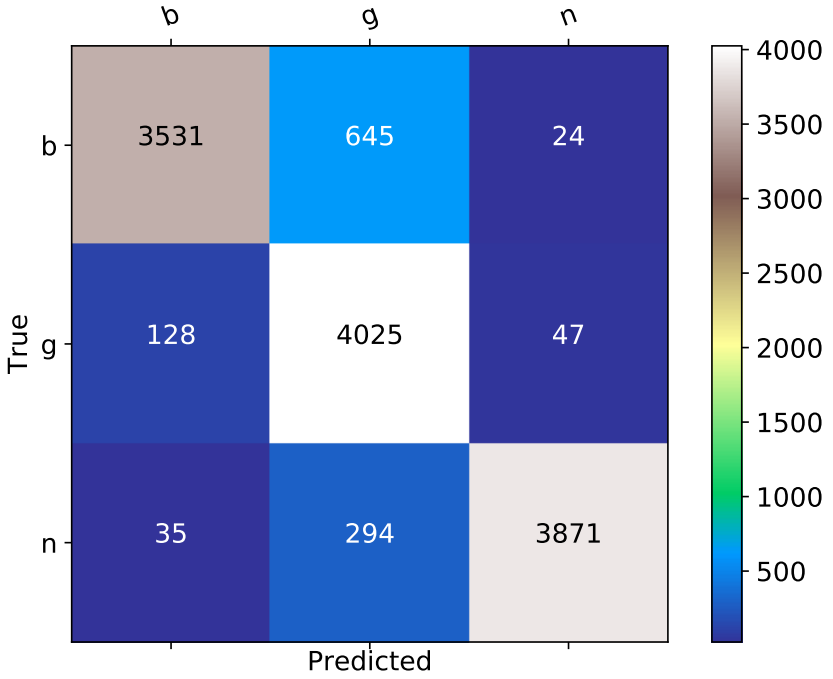


Figure 5.14. Confusion Matrix of 16-bit Grayscale Image Dataset.

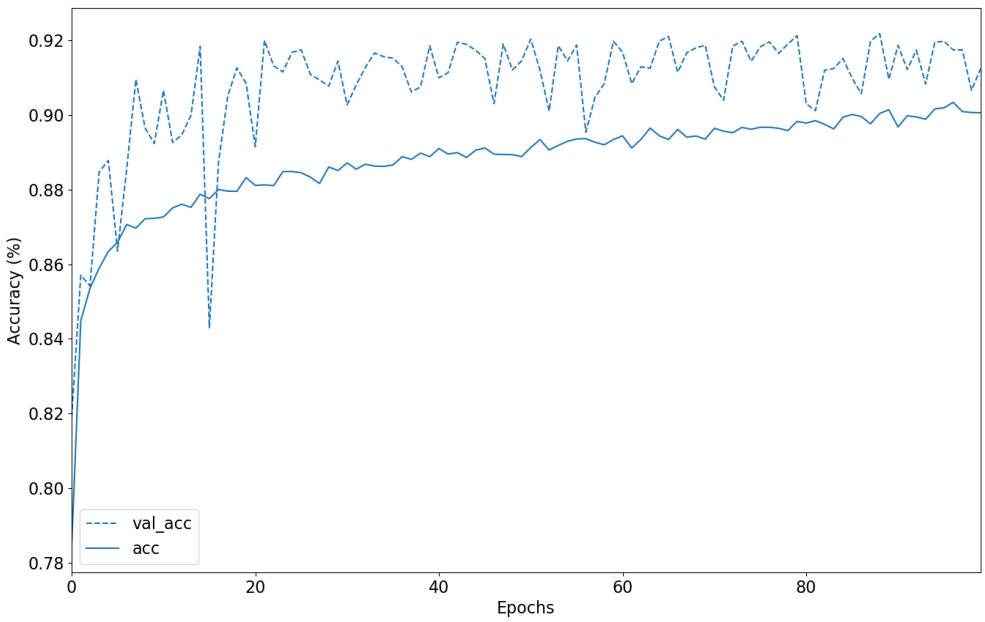


Figure 5.15. Accuracy vs Epochs for 16-bit Grayscale Image Dataset.

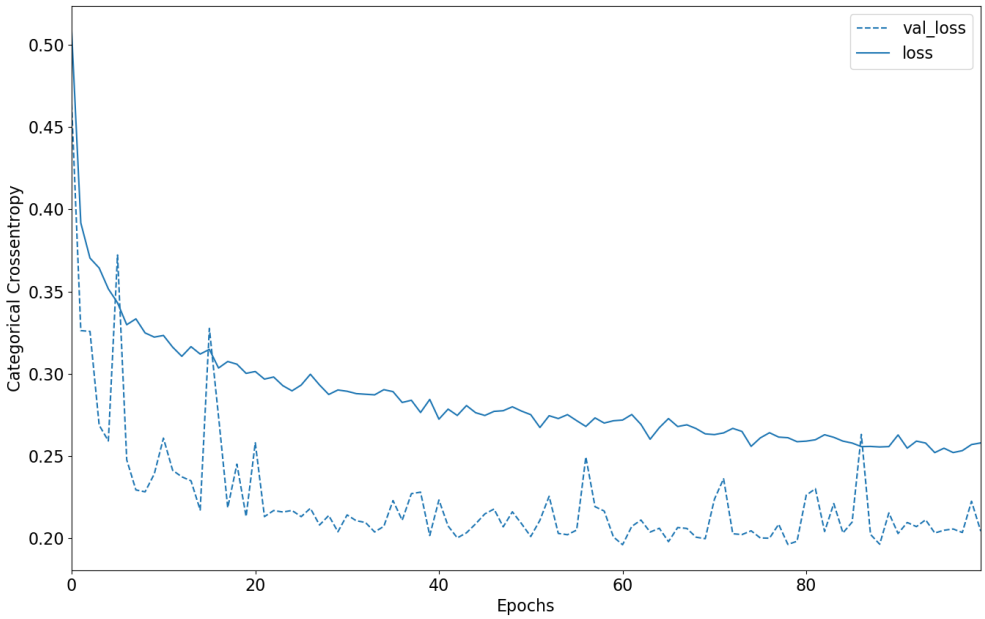


Figure 5.16. Loss vs Epochs for 16-bit Grayscale Image Dataset.

5.2. Real-time Tests on USRP E310

In this section, real-time test results on USRP E310 will be presented. After the all models are trained on a high-performance computer, they are tested on USRP E310. USRP E310 will achieve the prediction task with real-time RF samples. Designed CNN's software is written using Python language. Python language provides lots of frameworks, which are written by Google, Facebook, etc. for machine learning

Table 5.4. Grayscale vs RGB Training Comparison.

	8-bit Grayscale	**16-bit Grayscale**	**RGB**
Test Accuracy (%)	90.2	91.4	90.0
One Epoch Time (s)	24	25	30
Image Size (KB)	16	32	48

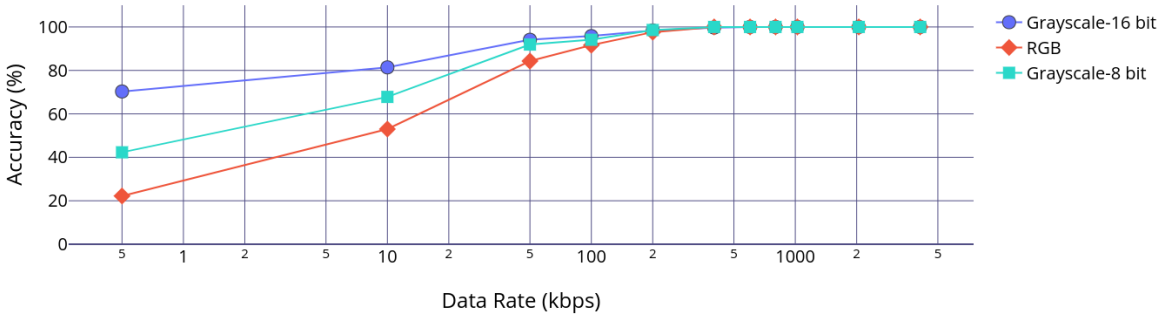


Figure 5.17. Accuracy vs Data Rate for 802.11b Signals.

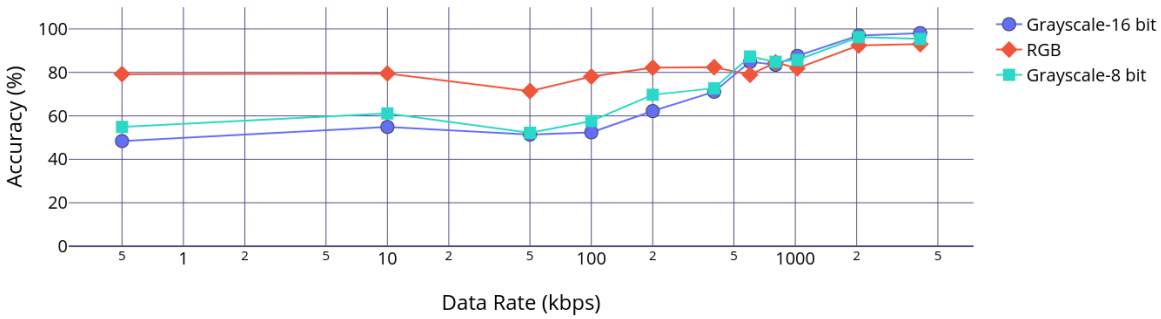


Figure 5.18. Accuracy vs Data Rate for 802.11g Signals.

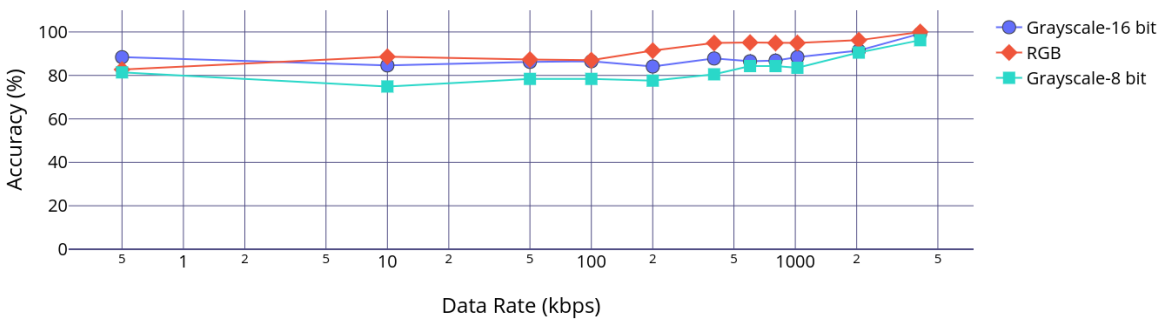


Figure 5.19. Accuracy vs Data Rate for 802.11n Signals.

and deep learning applications. These frameworks facilitate generating neural networks on the code. In this work, Tensorflow and Keras frameworks are employed for the designed CNN. However, these frameworks are available for high-end CPU architectures.

USRP E310 has ARM embedded CPU architecture for which these frameworks are not available. Therefore, the forward-propagation part of CNN is written from scratch using the mathematical library of Python named Numpy.

In real-time tests, raw IQ samples were collected for different data rates in the spectrum. Only one computer is connected to the wireless modem for the tests and random data is downloaded from the internet at different rates. Testing different rates will show the response of the CNN models to a variety of data rates. The data rate of the system in tests is set by a program called *wondershaper*. This program limits the network speed to a specific bit rate. Thus, the desired data rate in the spectrum is arranged. Collected raw samples are converted to RGB, 8-bit and 16-bit grayscale spectrograms. All the models are tested in real-time using these images.

Table 5.5. USRP E310 Performance on Real-Time Classification.

Task in CPU	**Duration (sec)**
Generation of One Spectrogram	2.6
Prediction of One Image	46.3

In Figures 5.17, 5.18 and 5.19, real-time test results for different data rates are given. The data rate is the amount of data that is transmitted by the wireless modem for the current test case, which is measured at the test computer and corresponds to the throughput of the connection between modem and computer. Therefore, this rate is not an actual bit rate in the spectrum. However, this will help in observing the data rate effect on the trained CNN model. The test case parameter is the true value of the current test and the prediction accuracies represent the probability of predicted values for each class. According to graphs, the grayscale model is better in 802.11b signals, the RGB model is a bit better in 802.11g, and the success rate of both models are approximately equal in 802.11n signals. Also, the 16-bit grayscale model is better than 8-bit in overall tests.

As one can observe from Table 5.5, the collection of RF samples and generation of spectrogram images are considerably faster compared to the prediction speed. The prediction of spectrograms using the CNN model is slow due to the CPU implementation of CNN. Also, note that USRP E310 has two CPU cores; therefore, two spectrogram images can be simultaneously predicted at the time it takes for predicting one spectrogram, effectively halving the prediction time per spectrogram.

6. CONCLUSION AND FUTURE WORK

The purpose of this thesis is designing hardware for 802.11b/g/n radio signal classification tasks. USRP E310 from Ettus Research is used as the hardware. Here, 802.11b/g/n signals are chosen as target signals, since collecting these signals are legal and they can be easily acquired from modems. This device can be employed in the military, emergency, and home applications thanks to its small size and standalone running capability. A computer connected USRP devices are employed in signal classification tasks in literature. The contribution of this thesis is using embedded standalone hardware for the classification task. This device achieves the classification of 802.11b/g/n signals. It is aimed to employ this device in real applications by improving its classification speed.

Three different CNN models are generated using 8-bit grayscale, 16-bit grayscale and 8-bit RGB images. The time-frequency information of the signal can be represented with the grayscale image, which only contains luminance information. RGB image increases complexity without making a considerable contribution to the signal information. The 16-bit grayscale model achieves better accuracy (91.4%) compared to RGB (90.0%). Grayscale is also superior in prediction time and image size. 16-bit grayscale model is also better than 8-bit (90.2%). One can employ the 16-bit grayscale model in 802.11b/g/n signal classification applications. The prediction performance of USRP is much slower because the classifier runs on the CPU.

Real-time predictions of signals on embedded CPU may not run fast enough for applications that have time limitations. If FPGA resources of USRP E310 would suffice, classifier could also be put on the FPGA side for better performance. Forward propagation part of CNN based classifier can be implemented in the FPGA as future work. This implementation will significantly increase speed of the predictions as well.

REFERENCES

1. Heath, J., *Nyquist rate basics and sufficient sampling for ADCs*, 2017, `https://www.analogictips.com/adcs-sufficient-sampling-nyquists-rate`, accessed in May 2019.

2. Matthew, B. and P. D. Shoemake, "Wi-Fi (IEEE 802.11 b) and Bluetooth coexistence issues and solutions for the 2.4 GHz ISM band", *Texas Instruments*, 2009.

3. Analog Devices, *PHY Basics: How OFDM Subcarriers Work*, `http://www.revolutionwifi.net/revolutionwifi/2015/3/how-ofdm-subcarriers-work`, accessed in June 2019.

4. Analog Devices, *802.11 OFDM Overview*, `http://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/wlan-ofdm/Content/ofdm_80211-overview.htm`, accessed in May 2019.

5. Neel Pandeya, N. T., *About USRP Bandwidths and Sampling Rates*, 2016, `https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates`, accessed in May 2019.

6. Ettus Research, *USRP^{TM} E310 Portable and Stand-alone*, 2019, `https://www.ettus.com/wp-content/uploads/2019/01/USRP_E310_Datasheet.pdf`, accessed in June 2019.

7. Haykin, S., D. J. Thomson and J. H. Reed, "Spectrum sensing for cognitive radio", *Proceedings of the IEEE*, Vol. 97, No. 5, pp. 849–877, 2009.

8. Kolodzy, P. and I. Avoidance, "Spectrum policy task force", *Federal Commun. Comm., Washington, DC, Rep. ET Docket*, Vol. 40, No. 4, pp. 147–158, 2002.

9. Yucek, T. and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications", *IEEE communications surveys & tutorials*, Vol. 11, No. 1, pp. 116–130, 2009.

10. Tandra, R., S. M. Mishra and A. Sahai, "What is a spectrum hole and what does it take to recognize one?", *Proceedings of the IEEE*, Vol. 97, No. 5, pp. 824–848, 2009.

11. Jagannath, J., H. M. Saarinen and A. L. Drozd, "Framework for automatic signal classification techniques (FACT) for software defined radios", *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pp. 1–7, IEEE, 2015.

12. Ferrus, R., O. Sallent, G. Baldini and L. Goratti, "Public safety communications: Enhancement through cognitive radio and spectrum sharing principles", *IEEE Vehicular Technology Magazine*, Vol. 7, No. 2, pp. 54–61, 2012.

13. Tallon, J., T. K. Forde and L. Doyle, "Dynamic spectrum access networks: Independent coalition formation", *IEEE Vehicular Technology Magazine*, Vol. 7, No. 2, pp. 69–76, 2012.

14. Joshi, G., S. Nam and S. Kim, "Cognitive radio wireless sensor networks: applications, challenges and research trends", *Sensors*, Vol. 13, No. 9, pp. 11196–11228, 2013.

15. Yucek, T. and H. Arslan, "Spectrum characterization for opportunistic cognitive radio systems", *MILCOM 2006-2006 IEEE Military Communications conference*, pp. 1–6, IEEE, 2006.

16. Hu, H., Y. Wang and J. Song, "Signal classification based on spectral correlation analysis and SVM in cognitive radio", *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, pp. 883–887, IEEE, 2008.

17. Tang, H., "Some physical layer issues of wide-band cognitive radio systems", *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, pp. 151–159, IEEE, 2005.

18. Palicot, J. and C. Roland, "A new concept for wireless reconfigurable receivers", *IEEE Communications Magazine*, Vol. 41, No. 7, pp. 124–132, 2003.

19. Jondral, F. K., "Software-defined radio: basics and evolution to cognitive radio", *EURASIP journal on wireless communications and networking*, Vol. 2005, No. 3, pp. 275–283, 2005.

20. Grayver, E., *Implementing software defined radio*, Springer Science & Business Media, 2012.

21. Ma, L., Z. Chen, L. Xu and Y. Yan, "Multimodal deep learning for solar radio burst classification", *Pattern Recognition*, Vol. 61, pp. 573–582, 2017.

22. O'Shea, T. J., T. Roy and T. C. Clancy, "Over-the-air deep learning based radio signal classification", *IEEE Journal of Selected Topics in Signal Processing*, Vol. 12, No. 1, pp. 168–179, 2018.

23. Mendis, G. J., J. Wei and A. Madanayake, "Deep learning-based automated modulation classification for cognitive radio", *2016 IEEE International Conference on Communication Systems (ICCS)*, pp. 1–6, IEEE, 2016.

24. Peng, S., H. Jiang, H. Wang, H. Alwageed and Y.-D. Yao, "Modulation classification using convolutional neural network based deep learning model", *2017 26th Wireless and Optical Communication Conference (WOCC)*, pp. 1–5, IEEE, 2017.

25. Topal, O. A., S. Gecgel, E. M. Eksioglu and G. K. Kurt, "Identification of Smart Jammers: Learning based Approaches Using Wavelet Representation", *arXiv preprint arXiv:1901.09424*, 2019.

26. Ettus, M. and M. Braun, "The universal software radio peripheral (usrp) family of low-cost sdrd", *Opportunistic Spectrum Sharing and White Space Access: The Practical Reality*, pp. 3–23, 2015.

27. Srivastava, S., M. Hashmi, S. Das and D. Barua, "Real-time blind spectrum sensing using USRP", *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 986–989, IEEE, 2015.

28. Atapattu, S., C. Tellambura and H. Jiang, "Energy detection based cooperative spectrum sensing in cognitive radio networks", *IEEE Transactions on wireless communications*, Vol. 10, No. 4, pp. 1232–1241, 2011.

29. Cabric, D., A. Tkachenko and R. W. Brodersen, "Experimental study of spectrum sensing based on energy detection and network cooperation", *Proceedings of the first international workshop on Technology and policy for accessing spectrum*, p. 12, ACM, 2006.

30. Sofotasios, P. C., E. Rebeiz, L. Zhang, T. A. Tsiftsis, D. Cabric and S. Freear, "Energy Detection Based Spectrum Sensing Over $\kappa-\mu$ and $\kappa-\mu$ Extreme Fading Channels", *IEEE Transactions on Vehicular Technology*, Vol. 62, No. 3, pp. 1031–1040, 2012.

31. Oh, D.-C. and Y.-H. Lee, "Energy detection based spectrum sensing for sensing error minimization in cognitive radio networks", *International Journal of Communication Networks and Information Security (IJCNIS)*, Vol. 1, No. 1, pp. 1–5, 2009.

32. Bhargavi, D. and C. R. Murthy, "Performance comparison of energy, matched-filter and cyclostationarity-based spectrum sensing", *2010 IEEE 11th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, IEEE, 2010.

33. Shobana, S., R. Saravanan and R. Muthaiah, "Matched filter based spectrum

sensing on cognitive radio for OFDM WLANs", *International Journal of Engineering and Technology*, Vol. 5, No. 1, pp. 142–146, 2013.

34. Kapoor, S., S. Rao and G. Singh, "Opportunistic spectrum sensing by employing matched filter in cognitive radio network", *2011 International Conference on Communication Systems and Network Technologies*, pp. 580–583, IEEE, 2011.

35. Huang, G. and J. K. Tugnait, "On cyclostationarity based spectrum sensing under uncertain Gaussian noise", *IEEE Transactions on Signal Processing*, Vol. 61, No. 8, pp. 2042–2054, 2013.

36. Tani, A. and R. Fantacci, "A low-complexity cyclostationary-based spectrum sensing for UWB and WiMAX coexistence with noise uncertainty", *IEEE Transactions on Vehicular Technology*, Vol. 59, No. 6, pp. 2940–2950, 2010.

37. Du, K.-L. and W. H. Mow, "Affordable cyclostationarity-based spectrum sensing for cognitive radio with smart antennas", *IEEE transactions on vehicular technology*, Vol. 59, No. 4, pp. 1877–1886, 2010.

38. Yucek, T. and H. Arslan, "OFDM signal identification and transmission parameter estimation for cognitive radio applications", *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, pp. 4056–4060, IEEE, 2007.

39. Rebeiz, E., F.-L. Yuan, P. Urriza, D. Marković and D. Cabric, "Energy-efficient processor for blind signal classification in cognitive radio networks", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 61, No. 2, pp. 587–599, 2013.

40. Xie, L. and Q. Wan, "Cyclic feature-based modulation recognition using compressive sensing", *IEEE Wireless Communications Letters*, Vol. 6, No. 3, pp. 402–405, 2017.

41. Swami, A. and B. M. Sadler, "Hierarchical digital modulation classification using

cumulants", *IEEE Transactions on communications*, Vol. 48, No. 3, pp. 416–429, 2000.

42. Wu, H.-C., M. Saquib and Z. Yun, "Novel automatic modulation classification using cumulant features for communications via multipath channels", *IEEE Transactions on Wireless Communications*, Vol. 7, No. 8, pp. 3098–3105, 2008.

43. Thilina, K. M., K. W. Choi, N. Saquib and E. Hossain, "Machine learning techniques for cooperative spectrum sensing in cognitive radio networks", *IEEE Journal on selected areas in communications*, Vol. 31, No. 11, pp. 2209–2221, 2013.

44. Park, C.-S., J.-H. Choi, S.-P. Nah, W. Jang and D. Y. Kim, "Automatic modulation recognition of digital signals using wavelet features and SVM", *2008 10th International Conference on Advanced Communication Technology*, Vol. 1, pp. 387–390, IEEE, 2008.

45. Joshi, H. and S. Darak, "Sub-Nyquist sampling and machine learning based online automatic modulation classifier for multi-carrier waveform", *2017 XXXIInd General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*, pp. 1–4, IEEE, 2017.

46. Boutte, D. and B. Santhanam, "A hybrid ICA-SVM approach to continuous phase modulation recognition", *IEEE Signal Processing Letters*, Vol. 16, No. 5, pp. 402–405, 2009.

47. Zhang, Z., Y. Li, X. Zhu and Y. Lin, "A method for modulation recognition based on entropy features and random forest", *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 243–246, IEEE, 2017.

48. Agarwal, A., H. Jain, R. Gangopadhyay and S. Debnath, "Hardware implementation of k-means clustering based spectrum sensing using usrp in a cognitive radio system", *2017 International Conference on Advances in Computing, Communi-*

cations and Informatics (ICACCI), pp. 1772–1777, IEEE, 2017.

49. Aslam, M. W., Z. Zhu and A. K. Nandi, "Automatic modulation classification using combination of genetic programming and KNN", *IEEE Transactions on wireless communications*, Vol. 11, No. 8, pp. 2742–2750, 2012.

50. Tang, B., Y. Tu, Z. Zhang and Y. Lin, "Digital signal modulation classification with data augmentation using generative adversarial nets in cognitive radio networks", *IEEE Access*, Vol. 6, pp. 15713–15722, 2018.

51. Jagannath, J., N. Polosky, D. O'Connor, L. N. Theagarajan, B. Sheaffer, S. Foulke and P. K. Varshney, "Artificial neural network based automatic modulation classification over a software defined radio testbed", *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.

52. Wu, H., Q. Wang, L. Zhou and J. Meng, "VHF radio signal modulation classification based on convolution neural networks", *Matec Web of Conferences*, Vol. 246, p. 03032, EDP Sciences, 2018.

53. Peng, S., H. Jiang, H. Wang, H. Alwageed, Y. Zhou, M. M. Sebdani and Y.-D. Yao, "Modulation classification based on signal constellation diagrams and deep learning", *IEEE transactions on neural networks and learning systems*, , No. 99, pp. 1–10, 2018.

54. Riyaz, S., K. Sankhe, S. Ioannidis and K. Chowdhury, "Deep learning convolutional neural networks for radio identification", *IEEE Communications Magazine*, Vol. 56, No. 9, pp. 146–152, 2018.

55. Zhang, M., M. Diao and L. Guo, "Convolutional neural networks for automatic cognitive radio waveform recognition", *IEEE Access*, Vol. 5, pp. 11074–11082, 2017.

56. Meng, F., P. Chen, L. Wu and X. Wang, "Automatic modulation classification:

A deep learning enabled approach", *IEEE Transactions on Vehicular Technology*, Vol. 67, No. 11, pp. 10760–10772, 2018.

57. Sadeghi, M. and E. G. Larsson, "Adversarial attacks on deep-learning based radio signal classification", *IEEE Wireless Communications Letters*, Vol. 8, No. 1, pp. 213–216, 2019.

58. Kokalj-Filipovic, S. and R. Miller, "Adversarial Examples in RF Deep Learning: Detection of the Attack and its Physical Robustness", *arXiv preprint arXiv:1902.06044*, 2019.

59. Bair, S., M. DelVecchio, B. Flowers, A. J. Michaels and W. C. Headley, "On the Limitations of Targeted Adversarial Evasion Attacks Against Deep Learning Enabled Modulation Recognition", *Proceedings of the ACM Workshop on Wireless Security and Machine Learning*, WiseML 2019, pp. 25–30, ACM, New York, NY, USA, 2019, `http://doi.acm.org/10.1145/3324921.3328785`.

60. Wireless Innovation Forum, *SDRF Cognitive Radio Definitions*, 2007, `http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf`, accessed in May 2019.

61. Machado-Fernández, J. R., "Software defined radio: Basic principles and applications", *Facultad de Ingeniería*, Vol. 24, No. 38, pp. 79–96, 2015.

62. Sadiku, M. N. and C. M. Akujuobi, "Software-defined radio: a brief overview", *Ieee Potentials*, Vol. 23, No. 4, pp. 14–15, 2004.

63. Kaur, S., C. Singh and A. S. Sappal, "Inter Carrier Interference Cancellation in OFDM System", *International Journal of Engineering Research and Applications (IJERA)*, Vol. 2, No. 3, pp. 2272–2275, 2012.

64. Tramarin, F., S. Vitturi, M. Luvisotto and A. Zanella, "On the use of IEEE 802.11 n for industrial communications", *IEEE Transactions on Industrial Informatics*,

Vol. 12, No. 5, pp. 1877–1886, 2015.

65. Ettus Research, *RFNoC*, 2018, https://kb.ettus.com/RFNoC, accessed in May 2019.

66. Braun, M., *What is GNU Radio?*, 2017, https://wiki.gnuradio.org/index.php/What_is_GNU_Radio\%3F, accessed in May 2019.

67. Ettus Research, *UHD*, 2017, https://kb.ettus.com/UHD, accessed in May 2019.

68. O'Shea, T. J., J. Corgan and T. C. Clancy, "Convolutional radio modulation recognition networks", *International conference on engineering applications of neural networks*, pp. 213–226, Springer, 2016.

69. Fehske, A., J. Gaeddert and J. H. Reed, "A new approach to signal classification using spectral correlation and neural networks", *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, pp. 144–150, IEEE, 2005.

70. Ettus Research, *E310/E312*, 2017, https://kb.ettus.com/E310/E312#Daughterboard_Specifications, accessed in May 2019.

71. Analog Devices, *RF Agile Transceiver*, 2016, https://www.analog.com/media/en/technical-documentation/data-sheets/AD9361.pdf, accessed in May 2019.

72. Collins, T., *AD9361, AD9364 and AD9363*, 2018, https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/ad9361, accessed in May 2019.

73. Analog Devices, *I/Q Correction*, 2016, https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms1-ebz/iq_correction, accessed in May 2019.

74. Flandrin, P., "Time–frequency filtering based on spectrogram zeros", *IEEE Signal*

Processing Letters, Vol. 22, No. 11, pp. 2137–2141, 2015.

75. Zhai, X., B. Jelfs, R. H. Chan and C. Tin, "Short latency hand movement classification based on surface EMG spectrogram with PCA", *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 327–330, IEEE, 2016.

76. Costa, Y. M., L. S. Oliveira and C. N. Silla Jr, "An evaluation of convolutional neural networks for music classification using spectrograms", *Applied soft computing*, Vol. 52, pp. 28–38, 2017.

77. Choudhary, T., L. Sharma and M. Bhuyan, "Spectracentrogram: A Time-Frequency Distribution for Signal Processing Applications", *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Vol. 3, pp. 1–5, IEEE, 2018.

78. Lu, W.-k. and Q. Zhang, "Deconvolutive short-time Fourier transform spectrogram", *IEEE Signal Processing Letters*, Vol. 16, No. 7, pp. 576–579, 2009.

79. Al Shalabi, L., Z. Shaaban and B. Kasasbeh, "Data mining: A preprocessing engine", *Journal of Computer Science*, Vol. 2, No. 9, pp. 735–739, 2006.

80. Han, J., J. Pei and M. Kamber, *Data mining: concepts and techniques*, Elsevier, 2011.

81. Fukushima, K., "Neocognitron: A hierarchical neural network capable of visual pattern recognition", *Neural networks*, Vol. 1, No. 2, pp. 119–130, 1988.

82. Zeiler, M. D. and R. Fergus, "Visualizing and understanding convolutional networks", *European conference on computer vision*, pp. 818–833, Springer, 2014.

83. Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information*

processing systems, pp. 91–99, 2015.

84. Li, H., Z. Lin, X. Shen, J. Brandt and G. Hua, "A convolutional neural network cascade for face detection", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5325–5334, 2015.

85. Cireşan, D., U. Meier and J. Schmidhuber, "Multi-column deep neural networks for image classification", *arXiv preprint arXiv:1202.2745*, 2012.

86. Piczak, K. J., "Environmental sound classification with convolutional neural networks", *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2015.

87. Merchant, K., S. Revay, G. Stantchev and B. Nousain, "Deep learning for RF device fingerprinting in cognitive communication networks", *IEEE Journal of Selected Topics in Signal Processing*, Vol. 12, No. 1, pp. 160–167, 2018.

88. Ioffe, S. and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *arXiv preprint arXiv:1502.03167*, 2015.

89. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

90. Khan, S., H. Rahmani, S. A. A. Shah and M. Bennamoun, "A guide to convolutional neural networks for computer vision", *Synthesis Lectures on Computer Vision*, Vol. 8, No. 1, pp. 1–207, 2018.

91. Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Advances in neural information processing systems*, pp. 1097–1105, 2012.

92. Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", *The Journal*

of Machine Learning Research, Vol. 15, No. 1, pp. 1929–1958, 2014.

93. Ramsundar, B. and R. B. Zadeh, *TensorFlow for deep learning: from linear regression to reinforcement learning*, " O'Reilly Media, Inc.", 2018.

94. LeCun, Y. A., L. Bottou, G. B. Orr and K.-R. Müller, "Efficient backprop", *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

95. Reitermanova, Z., "Data splitting", *WDS*, Vol. 10, pp. 31–36, 2010.

96. Aggarwal, C. C., *Neural networks and deep learning*, Springer, 2018.

97. Duchi, J., E. Hazan and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *Journal of Machine Learning Research*, Vol. 12, No. Jul, pp. 2121–2159, 2011.

98. Zhang, S., A. E. Choromanska and Y. LeCun, "Deep learning with elastic averaging SGD", *Advances in Neural Information Processing Systems*, pp. 685–693, 2015.

99. Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

100. Ruder, S., "An overview of gradient descent optimization algorithms", *arXiv preprint arXiv:1609.04747*, 2016.

101. Smith, S., *Digital signal processing: a practical guide for engineers and scientists*, Elsevier, 2013.