Waveform Optimization using Neural Network Based Real-Time Sensing in 5G Data Links

Final Year Project Report

By

Hadi Askari
Shaheer Mehmood

In Partial fulfilment
Of the requirements for the degree
Bachelors of Electrical Engineering (BEE)

School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan

(2021)

DECLARATION

We hereby affirm that this final year project titled as **Waveform Optimization using Neural Network based Real-Time Sensing in 5G Data Links** submitted to the School of Electrical Engineering and Computer Science, is a record of an original work done by us under the supervision and guidance of Prof. Dr. Faisal Shafait and Dr. Naveed Ahmed and that no part has been plagiarized without citations. This project work is submitted in the partial fulfillment of the requirements for the degree of Bachelors in Electrical Engineering.

Team Members

Shaheer Mehmood  Hadi Askari 

Supervisors

Prof. Dr. Faisal Shafait  Dr. Naveed Ahmed 

Date

May 11, 2021

DEDICATION

This thesis is dedicated to our parents, siblings, professors and mentors without whom, we would not have been here, and who have shown support and trust in us in all endeavors of our life. We also dedicate this thesis to our seniors for their encouragement that has enabled us to achieve everything in our lives.

ACKNOWLEDGMENTS

We are extremely grateful to our supervisor Dr. Faisal and especially our co-supervisor Dr. Naveed Ahmed for their continued support throughout the project. Their suggestion and feedback have allowed us to deliver on our work and this thesis would not have been possible had it not been for their mentorship and expert knowledge. We would also like to thank both of them for inspiring us to think outside the box and enabling us to become better problem solvers.

We would also like to acknowledge Dr Abdul Basit (NESCOM) for providing expertise regarding deep learning and neural networks and for providing us with real world data that allowed us to validate our model and improve its accuracy.

ABSTRACT:

A typical wireless channel has dynamic behavior in terms of noise, multipath fading and interference. The channel characterization in high-capacity data is a complex task. Conventional approaches target the worst-case channel impairments and adjusts the key design parameters accordingly, including prefix size, intercarrier distance, code rate, power level, sub-carrier modulation, transmit diversity and directionality. The availability of accurate state of wireless channel to a transmitter, especially in 5G and beyond data links, will enable the transmitter to optimize its waveform to achieve better average performance, namely improved energy efficiency and throughput. We will investigate new autonomous ways of channel sensing for 5G waveform using deep learning (DL) to discover channel state. The output of the DL model can be fed back to the transmit chain using the Channel State Information (CSI). This will enable the transmitter to adapt its waveform as per channel state to maximize link performance.

List of Figures

Contents

CHAPTER 1: INTRODUCTION

1.1 What is 5G

5G is considered as the next generation of wireless and telecommunication network technology that will enhance new strengths and transform the way we live our lives, perform daily tasks, interact with each other and play. 5G networks can be built in a variety of ways from a wide range of wavelength bands: low-band, mid-band, and high-band. High-band millimeter-wave frequencies have a large bandwidth that is there to support large amounts of data in densely populated urban areas but require cell sites to be compactly set and have limited access to buildings. Mid-band juggles with speed and range, providing wider coverage than high-band and less impacts by the buildings. In case of a low band, just like our powerful UHF and L - Band spectrum, travels farther than other bands - more than hundreds of square miles - and can go through many hurdles and obstacles, providing far better reach, coverage and better signal strength inside and out. There are many cases of use of 5G when it comes to modern times.

1.1.1 Developing Organizations:

5G opens the door to improved security and stability.

- Smart grids that significantly reduce carbon emissions
- Many connected vehicles share road safety information
- Prompt delivery in case of emergency situations
- Connected equipment and sensors which can not only detect but also inform before time when it comes to certain natural disasters
- Drones becomes an important tool for accelerating and supporting emergency response
- Remote technologies and specialists who consult / diagnose patients elsewhere

1.1.2 Industrial Transformation:

5G lays the foundation of business which promise responsibility, flexibility and efficiency.

- Production lines respond independently to demands and then their supply
- Digital duplicates that can warn of real machine errors ahead of time
- Logistic networks independently distribute goods according to real world conditions
- Full tracking for each item in warehouses and ports
- Remote access to certain robots and robust vehicles with improved and enhanced safety
- Increased use of certain IoT sensors and techniques in agriculture for the better growth of crops

1.1.3 Suggested Experience:

5G sets the stage for immersive entertainment and highly engaging education.

Great truth in AR, extended reality (XR) and VR and with relatively simpler devices and sensors.

- Bringing sensory sensations, such as touch, to devices
- Multi-participatory methods for teaching immersed content
- Immersive immersion sessions to increase remote product productivity
- Stable and reliable communication in densely populated areas
- New angles and viewer connections for live and long-distance events

The idea which makes 5G revolutionary is that it can not only provide data rates up to 20 Gbps which is 10 times more than 4G but also could provide massive network capacity by expanding beyond radio frequency spectrum and turning to mm Waves' spectrum. Thus, 5G promises overall uniform user experience by providing ultra-low latency, peak data rates and enhanced network capacities. Figure 1 illustrates

before us the evolution process undertaken by mobile communication in different decades.

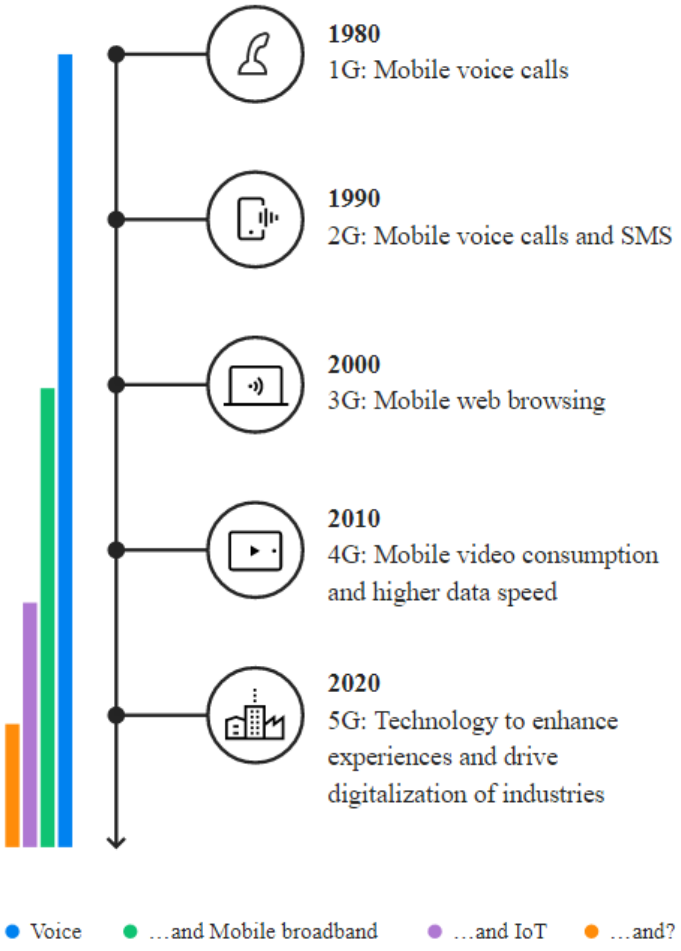

Figure 1 - Evolution of Mobile Communication

1.2 Digital Wireless communication

Wireless communication, despite the hype of the popular press, is a field that has been around for a very long-time span. The inception of this idea was around 1897 with was the era when Marconi's successful demonstrations regarding the process of wireless telegraphy were just emerging. By the year 1901, the concept of reception in radios across the Atlantic Ocean had been well established, illustrating that rapid progress in technology has also been around for quite a while. In the intervening

hundred years, many types of wireless systems have flourished, and often later disappeared (Gallager, 2006).

In digital wireless communication systems, information is transmitted through a radio channel.

In order to recover the actual information transmitted by certain conventional receivers, it is necessary to understand and decode the effect of the channel on the said signal. For instance, with binary phase shift keying (BPSK), binary information can be represented as +1 and −1 symbol values throughout the data. Now, inversion could take place inside the symbol values when the radio channel attempts to perform phase shift on the transmitted symbols. As long as the receiver can estimate what the channel did to the transmitted signal, it can accurately recover the information sent. Figure 2 represents the general flow of digital communication.

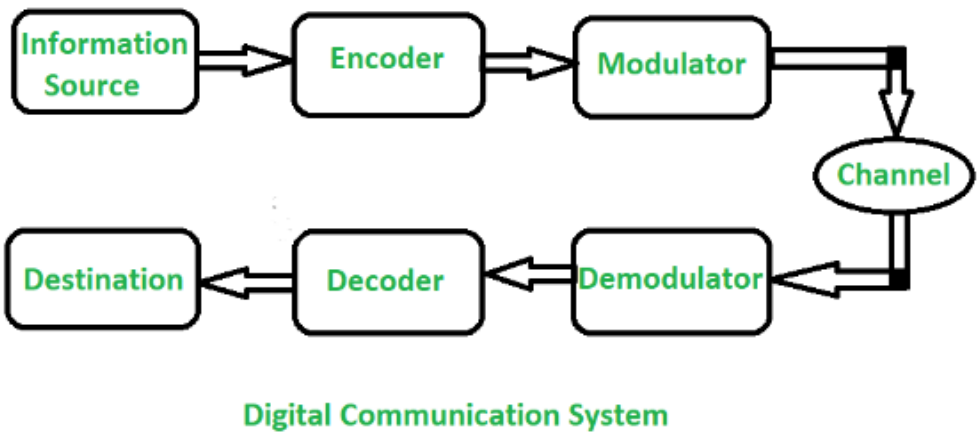

Figure 2 - Digital Communication

1.3 Channel Estimation

Channel estimation is considered a challenging problem when it comes to the functions involved in wireless communication. Transmission signals are usually displayed and spread, reaching the recipient in many ways. When these methods have the same delays, they add up constructively or destructively, resulting in blurring and distortions.

When these pathways have very different delays, they appear as signal echoes. Due to the transmission of the transmitter, receiver, or scattering material, the channel changes, and thus it must adapt over time.
The rated waves used in communication systems are interrupted by channel impairments such as Frequency Offset, Phase Offset, IQ Imbalance, AWGN, etc. Existing methods for identifying and removing these impairments from our received signal statistical methods based upon pilot techniques.

1.4 Motivation

Deep learning is a function of artificial intelligence that imitates the functionality and working procedure of a human brain when it comes to making patterns and processing data that could be further used in making certain decisions. We can consider it as a subset of machine learning in which our network is able to learn from unstructured data.
We propose a deep neural network-based method of identifying various channel impairments in 5G OFDM waves by creating a relative DNN model. Currently the Channel State information (CSI) is being calculated by pilot-based techniques which are of static nature. Our solution to calculate the CSI is by determining the impairments of a wave with the help of neural networks in real time and this information could be further used to estimate the channel.

1.5 Summary

This chapter provides us with the motivation behind our approach by discussing into the length the new 5G technology. It also helps us understand the basics of digital wireless communication and channel estimation in this very domain. Finally, this chapter closes with the motivation behind our solutions of channel estimation that are deep neural networks.

CHAPTER 2: LITERATURE REVIEW

There has been limited research conducted in this domain. Most of the related work is with regards to channel estimation and not impairment classification. There are several approaches used in the channel estimation realm that we took inspiration from.

2.1 Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems, 2017:

This was the renowned paper for deep learning for channel estimation and signal detection in OFDM systems. They used a deep learning approach to estimate the CSI implicitly and recover the transmitted data symbols directly. They implemented linear regression using a simple CNN model to minimize the bit error rate between the transmitted pilot symbols, hence estimating the channel. This was novel since it was varied from the already present receivers setup which estimated the CSI explicitly and after that recovered the transmitted symbols accordingly. The deep learning model was initially trained offline using the simulated data and later on it was used to recover the online transmitted data directly. This approach was much more robust than the already implied methods as relatively lesser training pilots were used, the cyclic prefix (CP) was removed and then the clipping noise which is nonlinear in nature was kept intact (Ye et al., 2018). Figure 3 (adapted from the same paper under discussion) shows the generalized system architecture.

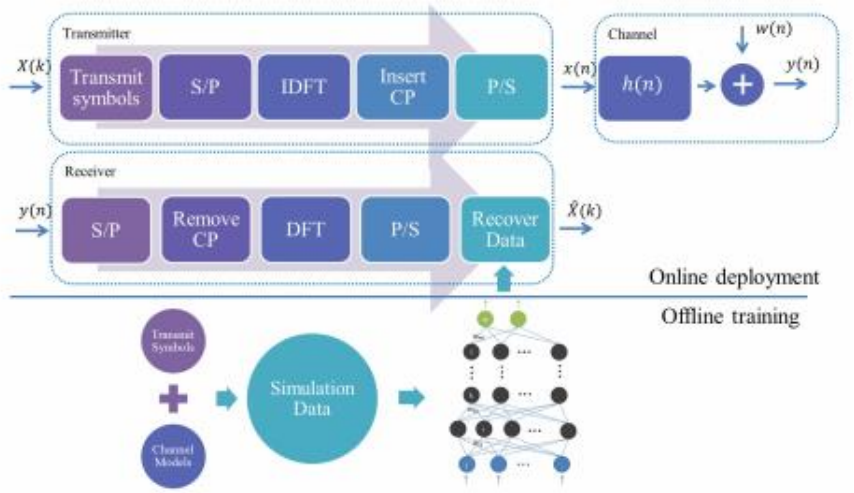

Figure 3- System Architecture

The transmitted symbols are first transformed into a paralleled data stream, then converted from the frequency to the time domain via Inverse Discrete Fourier Transform. A Cyclic Prefix is added to help with Inter-symbol Interference (ISI). The CP is later removed from the received signal and DFT is performed. A frame of the signal consists of the pilot symbols in the first OFDM block and the data in the succeeding OFDM blocks. The DNN model is trained by data of one pilot block and one data block and trained in an end-to-end fashion. In the offline portion of training, the model was trained with various OFDM samples with a myriad of information sequences and statistically variant channel properties. In the online portion, the DNN model generates the output that recovers the transmitted data without explicitly estimating the wireless channel (Ye et al., 2018).

Similar approaches were designed for doubly selective fading channels **(Deep learning-based channel estimation for doubly selective fading channels, 2019)** and for different IEEE standards **(Deep learning-based channel estimation schemes for IEEE 802.11P standard, 2020)** were also developed.

2.2 Detection Algorithms for Communication Systems Using Deep Learning, 2017

Many models built to detect the underlying communication channel between the transmitted and the received signal require accurate information of their underlying model. This work significantly showed that deep learning algorithms performed significantly better than the simple detectors in previous works. They performed experiments for both symbol by symbol and sequence detection. It was found that CNNs perform better for symbol-by-symbol detection and LSTMs performs better for Sequence Detection (Farsad & Goldsmith, 2017).

This work underlined the efficacy of Deep Learning Neural Networks in designing future communication systems.

2.3 An Introduction to Deep Learning for the Physical Layer, 2017

This was a major breakthrough in the development of our knowledge base. This paper was introduced to develop a new method of thinking regarding the functions in communications as a task which has an end-to-end reconstruction attribute. They used Auto-encoders to collectively understand the transmitter and receiver implementations along with the encodings present inside the signal without any previous knowledge. They modelled an entire communication channel, including the receiver and the transmitter. They optimized these by minimizing the block error rate (BLER). The entire transmitter, channel, receiver pipeline was conceptualized as one deep neural network and was trained as an auto-encoder.

An autoencoder down samples the input to a representation that has lesser dimensional at an internal layer, this is then reconstructed into the initial version at the output while minimizing the difference. Therefore the encoder non-linearly compresses and reconstructs the input. This is analogous to the case where you want to learn the representations of transmitted signals. We want a model that is robust to channel impairments so that it can be recovered with as few errors as possible (O'Shea & Hoydis, 2017).

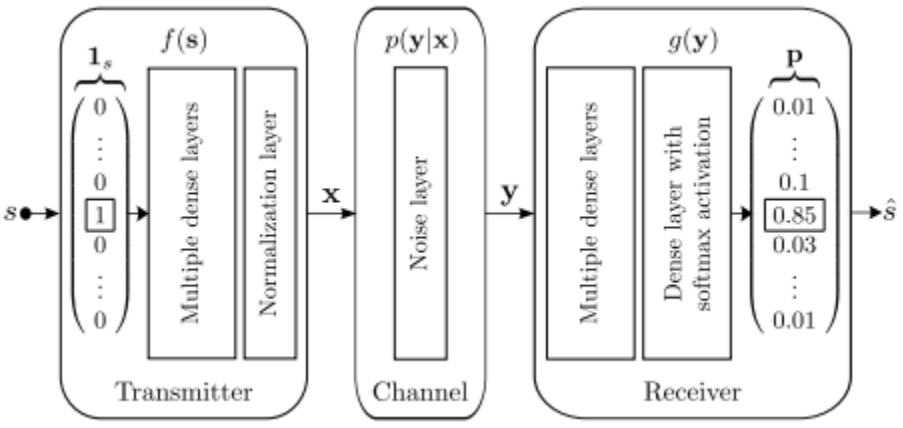

Figure 4 - Example of an Autoencoder

Figure 4 (adapted from the paper under discussion) consists of the modelling of a transmitter that consists of a feed-forward NN with multiple dense layers that is followed by a normalization layer ensuring that the physical specifications are met. The channel is represented by an additive white Gaussian noise layer with a fixed variance $\beta = (2RE_b/N_0)^{-1}$, where E_b/N_0 denotes the energy per bit (E_b) to noise power spectral density (N_0) ratio. The receiver is denoted by a feedforward NN where the last layer is a softmax layer whose output ranges from 0 to 1 over all of the possible messages. The autoencoder is then trained in an end-to-end fashion using SGD on a set of all possible messages using categorical cross entropy loss. The training was performed over a fixed value of E_b/N_o= 7dB using Adam optimizer with a learning rate of 0.001. The BLER was compared with a wide array of systems such as binary phase-shift keying (BPSK) modulation, a Hamming code with binary hard-decision coding and an uncoded BPSK. The autoencoder performed better in all scenarios. Similar experiments were performed for multiple transmitters and receivers with the benefits of autoencoders clearly being visible (O'Shea & Hoydis, 2017).

2.4 Radio Transformer Networks: Attention Models for Learning to Synchronize in Wireless Systems, 2016

This paper introduced radio transformer networks (RTNs) as a new method to integrate expert techniques and knowledge into the DL model. This performed well for modulation recognition for spatial transformer networks and had appropriate transformations according to the radio domain. Their main idea was to use attention as a way to further improve modulation classification. Traditionally, signals were synchronized before performing additional signal processing. The authors took inspiration and used this as a form of attention to estimate a time, phase, frequency and a sample timing offset to create a normalized version of the input. Attention models focus on a particular segment of the input data and remove variations to make downstream tasks easier and help in reducing their complexity (O'Shea et al., 2016).

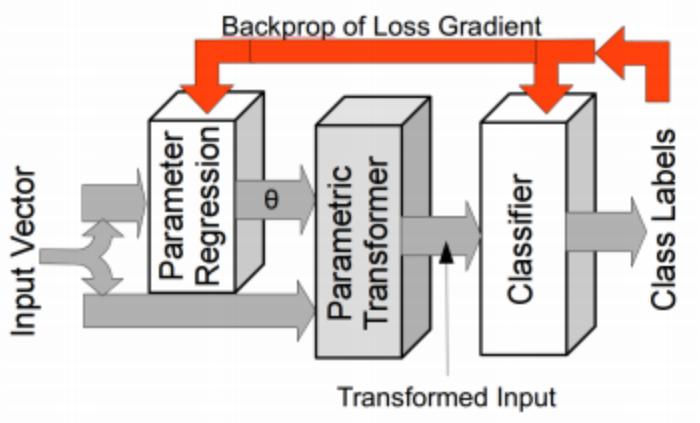

Figure 5 - Generalized Transformer Network Architecture

The Radio Transformer Network, such as one shown in Figure 5 (adapted from the research article under discusson), makes use of the generalization of the STN architecture while adding radio-domain specific parametric transforms. This shows encouraging signs in circumventing the computationally expensive analytical process and increasing the generalizability of machine learning based radio processing (O'Shea et al., 2016).

The authors limited channel variation to time offset, time dilation, frequency offset, and phase offset. They represented the data as a 2D image with two rows containing I/Q and N columns containing samples in time. They applied a mask to readily use 2D Affine transform implementations for translation and scaling, to recover phase and frequency they mix the signal with a complex sinusoid with the proper initial phase and frequency. This layer is cascaded before the Affine Transform in the Transformer Module. For the task of estimating these newly created parameters they add a Complex Convolution 1D Layer and convert from Complex Numbers to Power and Phase notation. The model was trained with trained via Adam optimization with a batch size of 1024 and an initial learning rate of 0.001(O'Shea et al., 2016). The model was trained for 350 epochs. Fig 6 shows a generalized RTN architecture and is adapted from this very piece of research paper.

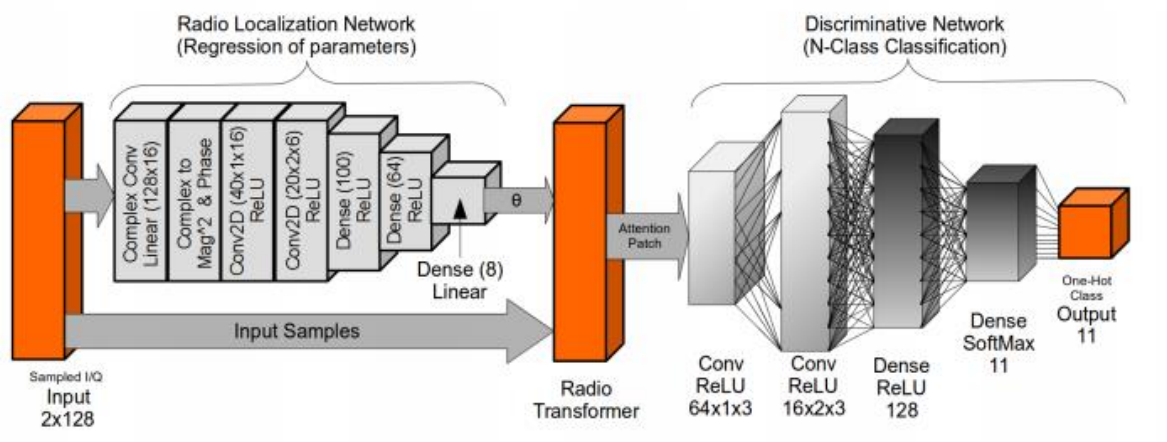

Figure 6 - RTN Architecture

2.5 Convolutional Radio Modulation Recognition Networks, 2016

This paper showed that modulation types and impairment noise levels can be classified using CNN models. This approach holds the potential to easily scale to additional modulation classes and should be considered as a strong candidate for Dynamic Spectrum Access (DSA) and Cognitive Radio (CR) systems which rely on robust low SNR classification of radio emitters. This work is the closest to our task as it is primarily involved with Modulation recognition (O'Shea et al., 2016).

The dataset is created in GNU Radio via the Radio channel model. 11 modulations were added to the dataset, 8 were digital and 3 analog. The samples are varied across a range of -20dB to a range of 20dB for an effective recreation of real-time scenarios. The simulations were exposed to different channel effects. Steps of 128 samples are extracted with a shift of 64 samples. Each sample is approximately 128

microseconds long and contains between 8 to 16 symbols with random time offset, scaling, rotation, channel response, noise and phase. The CNNs are trained to recover the information on how the channel was modulated by assigning one of a possible 11 class labels. Each corresponding to a possible modulation scheme. Training was performed on Keras using a 4 layer CNN with 2 convolutional layers. A dropout of 0.5 was used and the training included a categorical cross entropy loss function, optimized via Adam (O'Shea et al., 2016).

.

The results showed that the model was outperforming traditional feature based systems by 2.5-5dB of SNR. Confusion matrices were calculated for all the ranges of SNR and the best performance was found at higher SNRs.

2.6 Summary

By thoroughly going through the aforementioned literature and doing a comprehensive review, we highlighted and developed a mechanism to employ Machine Learning techniques for our problem in the radio domain. We learnt how to model the channel in an end-to-end fashion and how to interpret CSI to make decisions regarding the specifics of the model, without actually knowing the features of the channel. We developed several possible ways to tackle the problem of impairment classification for channel estimation including CNNs and RTNs and received several clues as to how to preprocess our complex-valued data for use in Neural Networks.

CHAPTER 3: PROBLEM STATEMENT

3.1 Problem Statement

The main motive is to create a model to identify the myriad of channel impairments that can arise during data transmission, after identification the next step is to actively remove these impairments with state-of-the-art accuracy while preserving the actual information sent. This is the main essence of the problem at hand.
For a given carrier wave, we need to design a model that can robustly and in real-time, identify the impairments that are in the wave, along with the levels of those impairments. We want to perform this task implicitly using the channel state information (CSI) so that waveform correction can be applied at the receiver end. This entails three major steps in our project timeline and objectives.

- Our first objective was to understand how to model and simulate the different impairments. We performed the visualization, simulation, and further preprocessing using MATLAB 5G New Radio Toolbox. We had to write MATLAB scripts on how to extract the important information from the waves. After recreation and extraction, we will create a classification model to identify the impairments in the waves.
- The input 5G waves will then be fed into multiple DL models to identify what types and levels of impairments are present in the waves. There are several candidates to the specifics of this DL model, we will experiment amongst CNNs and Radio Transformer Networks to identify what will work best with our model.
- Finally, we will create a robust model capable of real time detection and removal of these channel impairments. We intend to create a Python script where a user can input a wave upon a certain carrier frequency and then the model can return the impairment profile of the 5G wave.

CHAPTER 4: METHODOLOGY

Figure 7 shows the basic methodology followed in order to proceed with our project.

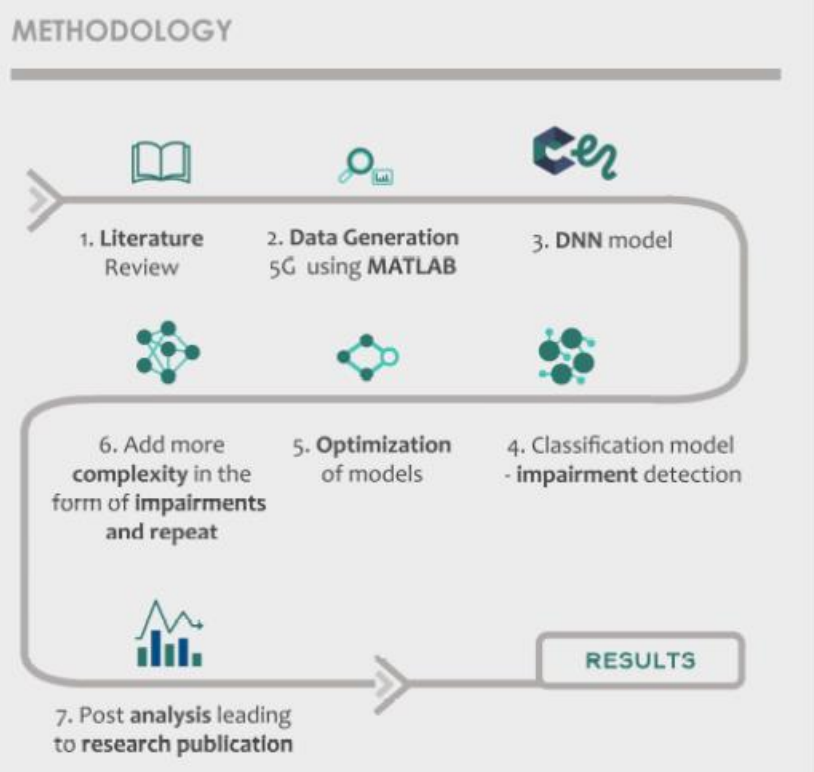

Figure 7 - Basic Methodology of our project

The three main modules in our project are as follows:

- Data Generation
- Data Pre processing
- Deep Learning Model

4.1 Data Generation:

The first module in our model was to generate an ample of data on 5G waves that could ultimately be used for our deep learning model. Data on 5G waves isn't available, thus one had to resort to MATLAB for this module. MATLAB 2020a version provides us a Waveform Generator app that helps us to generate 5G waves. This app also gives us flexibility to decide the bandwidth, modulation scheme, sub caring spacings, Cell ID and types of impairments introduced.

The basic interface of 5G Waveform Generator is as follows in Figure 8

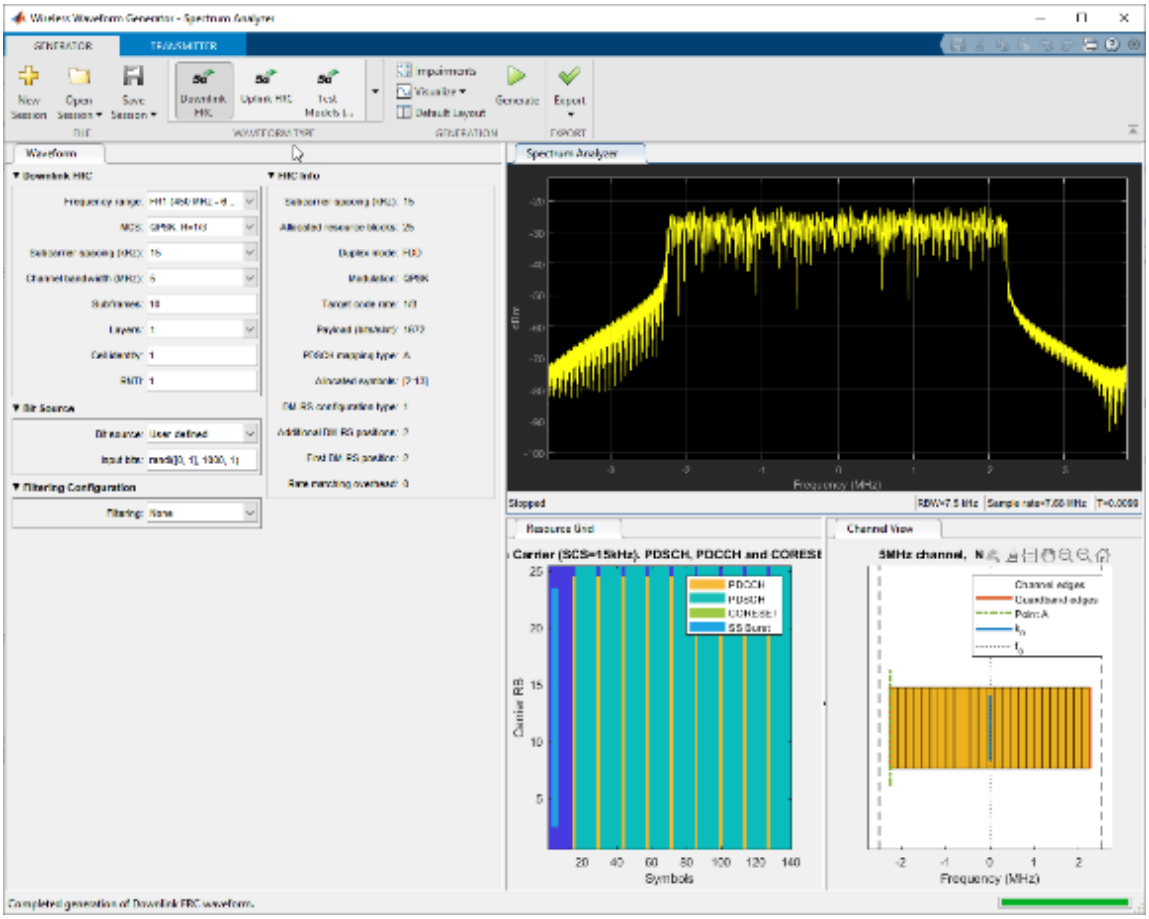

Figure 8 - 5G Waveform Generator Interface (MATLAB 2020a)

4.1.1 Specifications:

The specification that was decide for our 5G OFDM waves were as follow

- Frequency Range = FR1
- Bandwidth = 5MHz
- Modulation Scheme = QPSK
- Target Code Rate = 1/3
- Bit Source = User Defined (For random wave generation)
- Cell Identity = 1
- Subframes = 1000 (could be adjusted depending upon the amount of Data required)

Thus, when you are generating waves with the help of a 5G Toolbox in MATLAB 2020a, you need to set these following specifications. The first parameter that is needed is the frequency range. 5G waves in MATLAB have two frequency ranges, that are FR1 and FR2. FR1 refers to the frequency range that is currently used for all digital communication and 5G will also use it while FR2 refers to microwaves frequencies which are inhabited at the moment and could serve specifically for 5G communication. We started with the base case and opted for FR1 for our data generation.

Frequency Range Designation	**Corresponding Frequency Range**
FR1	450 MHz – 6000 MHz
FR2	24250 MHz – 52600 MHz

The bandwidth of 5G waves keeps on varying. The demodulates at the receiver ends always have a knowledge of the bandwidth of the transmitted wave. As our project is research based, we plan to start with just 5MHz waves and study their CSI and later one move to other cases and waves with varying bandwidths.

Modulation Scheme and Target coding can vary from one wave to another. Depending upon the nature of transmitted data and the transmitter, both of these parameters change. In our project, we opted for a general case and chose the modulation scheme as QPSK and target coding as 1/3.

The utility of defining Bit Source as user defines is that every time the 5G Toolbox generates a wave, the data that specific wave carries will be randomly generated. This feature helps us get a data that is quite realistic in nature and the deep learning results generated on the basis on this data could also easily account for real time 5G communication because of similarity in the rate of 5G waves data.

Cell Identity or commonly called Cell ID is actually the number of cells that are in the neighborhood of our cell in a cluster. This Cell ID could vary from 1-3 depending upon the target neighbor. In our case, we chose Cell ID as 1 (as a base case). If one needs to generate more information out of a single wave, varying the cell ID number could be of help.

Finally, every 5G wave has a data frame and ultimately subframes. The number of subframes ultimately define how much data a wave is carrying and consequently decide the size of our 5G wave. More is the number of subframes, this could mean more channels in a wave and ultimately more frames. In our case, we kept on changing the number of subframes to change the amount of data we need out of a single 5G wave.

4.1.2 Impairments:

As the main task of our project was to identity the levels and the types of impairments in our wave, thus during the process of data generation we decided to go with 3 basic impairments that are dominant when it comes to wireless communication in 5G OFDM waves. The impairments are as follows:

- AWGN (Additive White Gaussian Noise)
- Frequency Offset
- Path Delays

AWGN

Additive white Gaussian sound (AWGN) is the basic sound source used in the concept of knowledge to mimic the effect of many random processes occurring in nature. Modifiers explain certain features:

- Additive because it is included in any sound that can be part of the information system.
- White implies that it has the same power over every common band of information system. It is the same color as white with the same exposure to all the wavelengths.
- Gaussian because it has a standard distribution in the time zone at a time value of zero time

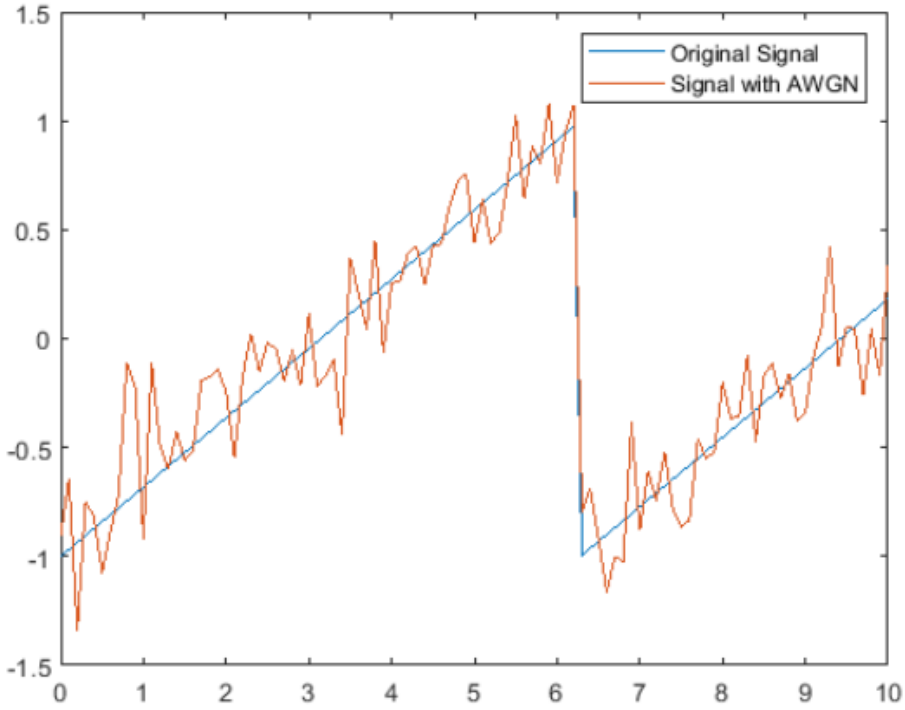

Figure 9 - Additive White Gaussian Noise effects to a wave

Range: 0 – 30db

MATLAB function/Channel used: comm.AWGNChannel

Frequency Offset:

Frequency offset of the network occurs when the local oscillator signal to switch down the receiver does not match the network signal contained in the received signal. This condition can be caused by two important factors: a mismatch of the frequency in the transmitter and the receiver oscillator; and the Doppler effect as the transmitter or receiver moves.

If this happens, the received signal will be moved several times. In the OFDM system, orthogonality between subcarriers is maintained only if the receiver uses a local oscillation signal that is compatible with the network signal contained in the received signal. Alternatively, incompatibility with network traffic can lead to inter-carrier (ICI) disruption. The oscillators on the transmitter and receiver cannot be attracted to the same frequencies. Therefore, the carrier frequency of the network company remains with or without the effect of Doppler Figure 10 provides visual aid to the above-mentioned discussion.

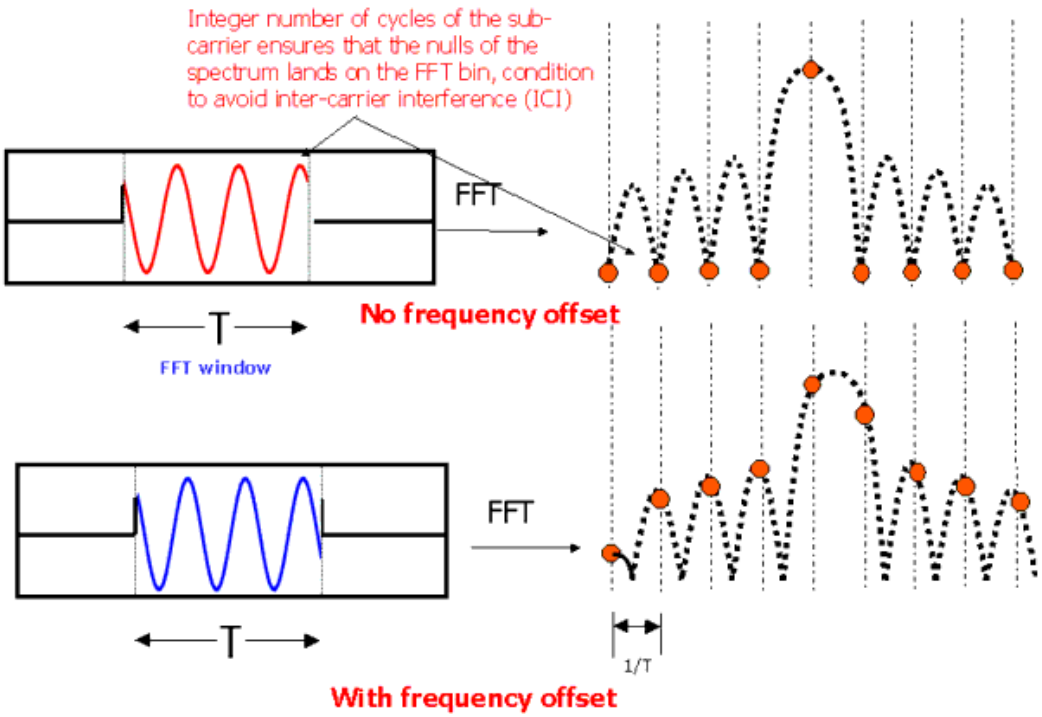

Figure 10 - Frequency Offset effects on an OFDM wave

Range: 0 – 500kHz

MATLAB function/Channel used: comm.PhaseFrequencyOffset

Path Delays:

Multipaths inside the radio channel causes the creation of certain small-scale fading effects. We have tried to discuss only the three most import effects as follows:

- Instant changes in the signal strength over a relatively smaller traveling distance or time interval
- Random frequency modulation which is cased due to varying Doppler shifts on various other multipaths
- Time dispersion / echoes which is caused by multipath propagation delays.

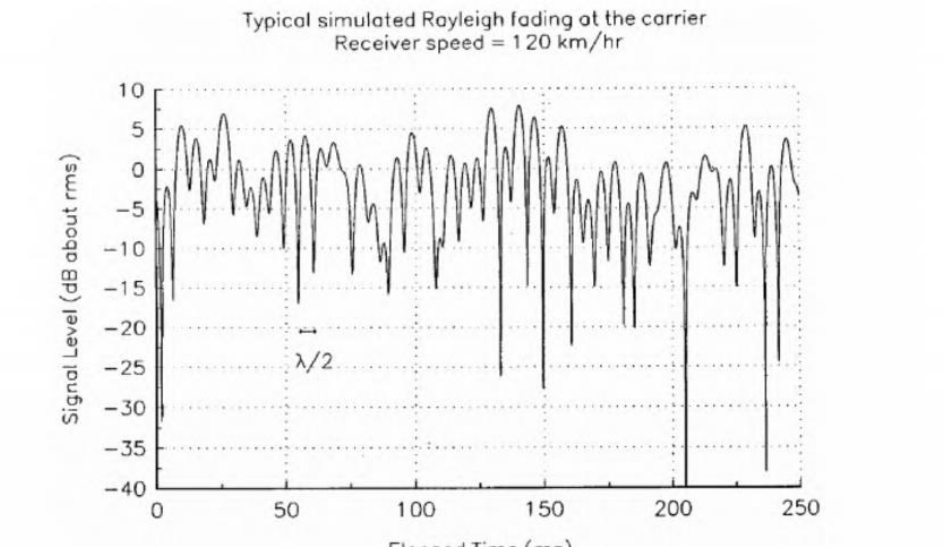

Figure 11 - Path Delays illustration in an OFDM wave

Range: 0 – 30us

MATLAB function/Channel used: comm.RayleighChannel

4.1.3 Data Set:

In order to classify the impairment type along with the levels, we decided to make three different CNN classifiers for each impairment that is AWGN, Path Delays and Frequency Offset. Thus, for three classifier we had to make three different data sets:

- Variable Noise data has the worst-case impairments of frequency offset and Path Delays while a varying level of AWGN
  AWGN: 0 – 30 dB
  Frequency Offset: 6 kHz
  Path Delays: 30 us

- Variable Frequency data has the worst-case impairments of AWGN and Path Delays while a varying level of frequency offset
  AWGN: 30 dB
  Frequency Offset: 0 – 6 kHz
  Path Delays: 30 us

- Variable Path Delays data has the worst-case impairments of frequency offset and AWGN while a varying level of path delays
  AWGN: 30 dB
  Frequency Offset: 6 kHz
  Path Delays: 0 – 30 us

4.2 Data Pre-Processing:

In order to understand the choice of extraction of PSS-1 (Primary Synchronous Signal 1) from the entire 5G – OFDM wave, it is very important to understand the utility, importance, location and the repeatability of the SS – Block (Synchronous Signal) in the 5G data frame.

4.2.1 The SS – Block:

Synchronization is part of the physical layer, backbone of any new wireless technology, with which evolution moves on together with the available technology and requirements. Looking at the mobile standards history, the synchronization issue has always been studied, producing many different methods, often with different targets and requirements.

In LTE systems two special signals and one physical channel have been defined to perform synchronization. They are the Primary Synchronization Signal (PSS), Secondary Synchronization Signal (SSS) and a Physical Broadcast Channel (PBCH) with its Demodulation Reference Signal (DMRS), jointly indicated as 5G – OFDM PSS/SSS/PBCH.

As shown in Figure 12, 5G takes LTE PSS/SSS/PBCH to build a single structure called Synchronization Signal Block (SSB), periodically transmitted on the downlink by each NR cell to perform the synchronization procedure: devices can understand they are inside the coverage area of a cell by finding the SS-Block. It occupies a total amount of 960 resource elements in the OFDM grid: 4 OFDM symbols in time domain and 240 contiguous subcarriers (20 RBs) in the frequency domain, for each symbol. The actual band in Hz and time duration in sec of the SS-Block depends on the numerology adopted by the network.

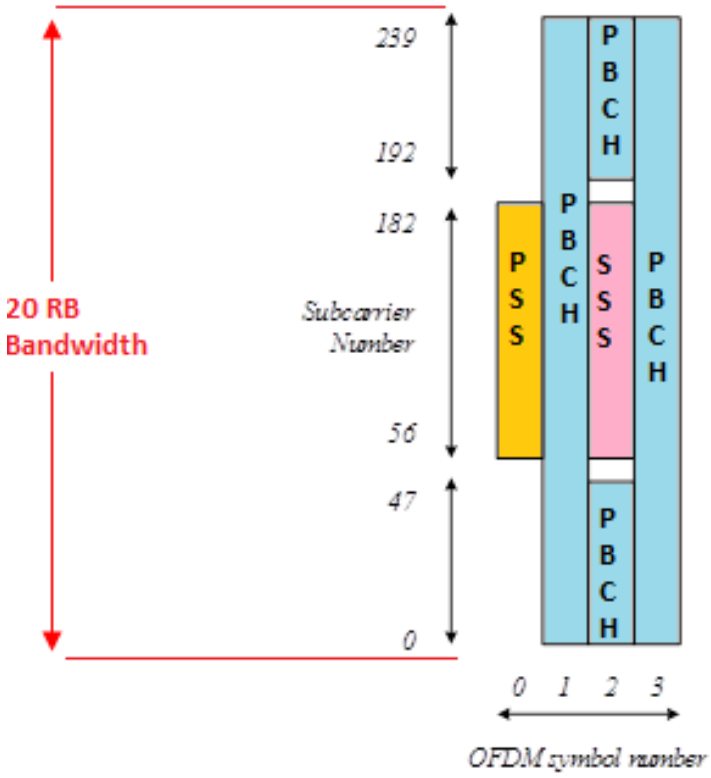

Figure 12 - 5G OFDM Grid

4.2.2 Primary Synchronous Signal (PSS):

The Primary and Secondary synchronization signals permit the determination of the Physical Cell Identity (PCID) of the cell. In 5G – NR radio interface PCID can assume 1008 different values. The PSS sequence is the first synchronization signal inside the received signal and also the first signal the UE (User Equipment) searches for.

PSS purpose is to define the initial symbol alignment and apply a coarse frequency correction, since at initial access stage the system timing is not available and the UE oscillator inaccuracy could produce a mismatch with the system reference carrier frequency.

4.2.3 Preprocessing Approach

A 5G OFDM grid has a lot of channel that have varied functionality. To discuss the functionality of all the channels is out of the scope of this project. We only plan to target the PSS-1 that is the Primary Synchronous Signal 1. A wave (such as one shown in Figure 13) has multiple PSS and it repeats after every 10 subframes. Each subframe has 14 OFDM symbols and PSS-1 (which is our desired PSS) is located in the 3rd symbol. Thus, knowing this information and the bandwidth, we came to know the size of each symbol. This allowed us to write a MATLAB script that would extract all the PSS – 1 sample form the entire wave.

Extracting the PSS (as shown in Figure 14) is also necessary as a 5G wave's length is quite huge and one could not use this entire waveform comprising of millions of complex doubles as an input for a DNN (specifically CNNs and RTNs), thus we had to apply respective preprocessing to it. We finally decided to extract out the PSS – 1.

4.2.4 Novelty

The prior approach to apply deep neural networks (DNNs) to digital wireless communication problems such as modulation type classification (Ref) required to use the entire LTE waveform to be used as input for the neural network. This process was inefficient in the sense that the network had to process the entire wave in the training phase which not only cost resources but also time.

Later on, another approach of auto encoder (Ref) was introduced in which the OFDM wave was scaled downed to just the important features using deep learning techniques. The results produced with this method were better than the previous approaches but at time, the auto encoder gave weightage to the less important features of the wave.

Extraction of PSS (in our case the PSS-1) proves to be an authentic method as PSS does not only repeat after every 10 subframes, enhancing its credibility, but also carries the information regarding the specification of the wave and its symbols alignment. Thus, a 5G OFDM wave could be well illustrated by its PSS and we could further use it as a representation for the entire wave.

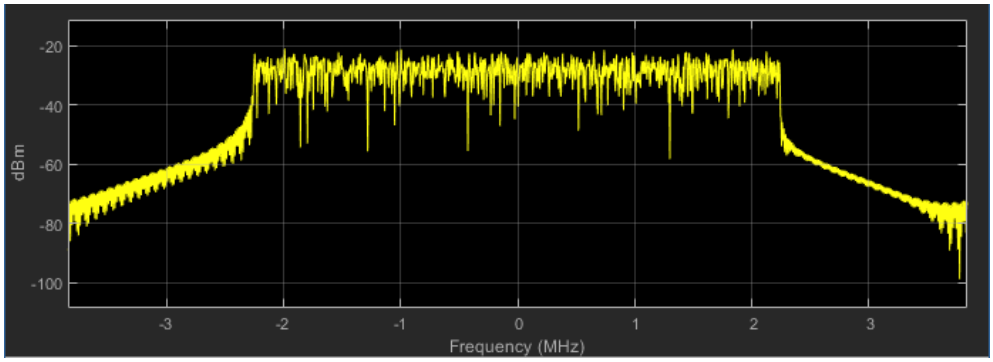

Figure 13 - 5G wave with impairment produced via 5G waveform generator MATLAB 2020a

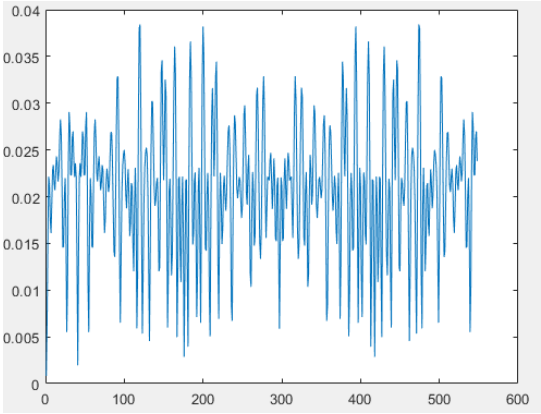

Figure 14 – Extracted PSS 1 (1 in every 10 subframes)

4.3 ML Architecture

We have experimented with several architectures in our research. We considered traditional Machine Learning methods, autoencoders, LSTMs, RNNs, CNNs and RTNs. Eventually, we settled on CNNs to be the basis of our implementation as they should the most promise with handling our impairment classification and are diverse and easy to use enough to comfortably generalize to our given task at hand. The following are the two main models we used to conduct our experiments.

4.3.1 CNN - 4

```
dr = 0.5
model = Sequential()
model.add(Reshape(out_shape, input_shape = in_shape))
model.add(ZeroPadding2D((0, 2), data_format = 'channels_first'))
model.add(Conv2D(256, (1, 3), padding = 'valid', activation = "relu", name="conv1", kernel_initializer='glorot_uniform', data_format="channels_first"))
model.add(Dropout(dr))
model.add(ZeroPadding2D((0,2), data_format = 'channels_first'))
model.add(Conv2D(80, (2, 3), activation="relu", name="conv3", padding="valid", kernel_initializer="glorot_uniform", data_format="channels_first"))
model.add(Dropout(dr))
model.add(Flatten())
model.add(Dense(256, activation="relu", name="dense1", kernel_initializer="he_normal"))
model.add(Dropout(dr))
model.add(Dense(6, name="dense3", kernel_initializer="he_normal", activation = 'softmax'))
model.add(Reshape([len(list_labels)]))
```

Figure 15 - 4 Layer CNN Architecture

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_2 (Reshape)          (None, 1, 2, 548)         0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 1, 2, 552)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 256, 2, 550)       1024
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 256, 2, 554)       0
_________________________________________________________________
conv3 (Conv2D)               (None, 80, 1, 552)        122960
_________________________________________________________________
flatten_1 (Flatten)          (None, 44160)             0
_________________________________________________________________
dense1 (Dense)               (None, 256)               11305216
_________________________________________________________________
dense3 (Dense)               (None, 5)                 1285
_________________________________________________________________
reshape_3 (Reshape)          (None, 5)                 0
=================================================================
Total params: 11,430,485
Trainable params: 11,430,485
Non-trainable params: 0
_________________________________________________________________
```

Figure 16 - Model Parameter (4-layer CNN)

The model in Figure 15 and Figure 16 is a simple CNN architecture. It takes an input shape of (1, 2, 548). Then we applied zero padding to ensure that the convolution output feature maps are to our specifications. This is of the format channels_first which corresponds to inputs with shape (batch_size, channels, height, width). Next we add a convolutional layer that has 256 filters and a kernel size of (1, 3). We have already applied zero padding hence we are keeping it as 'valid' which means that the model will apply no padding. We used the activation function 'relu' and used the glorot initializer for kernel initialization. Following another zero padding layer we add another convolutional layer of 80 filters that has a kernel size of (2,3) with the remaining specifications as before. We then use a

Fully Connected layer with 256 neurons, with ‘He_normal’ kernel initialization and the ‘relu’ activation function. Next, we add dropout of 0.5 to ensure that the model trains properly. Finally, we have a softmax layer of 6 output neurons corresponding to our 6 noise levels.

The loss function we are using is 'categorical_crossentropy' and we are using ‘adam’ optimizer for optimization. We will calculate train accuracy of our model and validation accuracy.

4.3.2 CNN - 17

```
[22] model = Sequential()
     model.add(Reshape(out_shape, input_shape = in_shape))
     model.add(ZeroPadding2D((2, 2), data_format = 'channels_first'))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(16, kernel_size=(3,3), activation='relu', strides=(1, 1)))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(16, kernel_size=(3,3), activation='relu', strides=(1, 1)))
     model.add(BatchNormalization())
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(32, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.05)))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(32, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.05)))
     model.add(BatchNormalization())
     model.add(Conv2D(64, kernel_size=(1,1), activation='relu',strides=(1, 1), padding='valid'))
     model.add(Conv2D(64, kernel_size=(1,1), activation='relu',strides=(1, 1), padding='valid'))
     model.add(BatchNormalization())
     model.add(MaxPooling2D(pool_size=(1, 1)))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(128, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.01)))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(128, kernel_size=(2,2), activation='relu', strides=(1, 1), padding='valid'))
     model.add(BatchNormalization())
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(64, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.02)))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(64, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.02)))
     model.add(BatchNormalization())
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(64, kernel_size=(3,3), activation='relu', strides=(1, 1)))
     model.add(ZeroPadding2D(padding=(1, 1)))
     model.add(Conv2D(64, kernel_size=(3,3), activation='relu', strides=(1, 1)))
     model.add(BatchNormalization())
     model.add(ZeroPadding2D(padding=(1, 1)))
```

Figure 17 - 17 Layer CNN Architecture (1)

```
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.02)))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.02)))
model.add(BatchNormalization())
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(128, kernel_size=(3,3), activation='relu', padding='valid', kernel_regularizer=regularizers.l2(0.05)))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(128, kernel_size=(2,2), activation='relu', strides=(1, 1), padding='valid'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(32, activation="relu", name="dense1", kernel_initializer="he_normal",))
model.add(Dropout(0.3))
model.add(Dense(32, activation="relu", name="dense2", kernel_initializer="he_normal",))
model.add(Dropout(0.2))
model.add(Dense(6, activation='softmax'))
model.add(Reshape([len(list_label)]))
```

Figure 18 - 17 Layer CNN Architecture (2)

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_2 (Reshape)          (None, 1, 2, 548)         0
_________________________________________________________________
zero_padding2d_16 (ZeroPaddi (None, 1, 6, 552)         0
_________________________________________________________________
zero_padding2d_17 (ZeroPaddi (None, 3, 8, 552)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 1, 6, 16)          79504
_________________________________________________________________
zero_padding2d_18 (ZeroPaddi (None, 3, 8, 16)          0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 1, 6, 16)          2320
_________________________________________________________________
batch_normalization_8 (Batch (None, 1, 6, 16)          64
_________________________________________________________________
zero_padding2d_19 (ZeroPaddi (None, 3, 8, 16)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 1, 6, 32)          4640
_________________________________________________________________
zero_padding2d_20 (ZeroPaddi (None, 3, 8, 32)          0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 1, 6, 32)          9248
_________________________________________________________________
batch_normalization_9 (Batch (None, 1, 6, 32)          128
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 1, 6, 64)          2112
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 1, 6, 64)          4160
_________________________________________________________________
batch_normalization_10 (Batc (None, 1, 6, 64)          256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 6, 64)          0
```

Figure 19 - Model Parameter (17-layer CNN) (1)

zero_padding2d_21 (ZeroPaddi	(None, 3, 8, 64)	0
conv2d_22 (Conv2D)	(None, 1, 6, 128)	73856
zero_padding2d_22 (ZeroPaddi	(None, 3, 8, 128)	0
conv2d_23 (Conv2D)	(None, 2, 7, 128)	65664
batch_normalization_11 (Batc	(None, 2, 7, 128)	512
zero_padding2d_23 (ZeroPaddi	(None, 4, 9, 128)	0
conv2d_24 (Conv2D)	(None, 2, 7, 64)	73792
zero_padding2d_24 (ZeroPaddi	(None, 4, 9, 64)	0
conv2d_25 (Conv2D)	(None, 2, 7, 64)	36928
batch_normalization_12 (Batc	(None, 2, 7, 64)	256
zero_padding2d_25 (ZeroPaddi	(None, 4, 9, 64)	0
conv2d_26 (Conv2D)	(None, 2, 7, 64)	36928
zero_padding2d_26 (ZeroPaddi	(None, 4, 9, 64)	0
conv2d_27 (Conv2D)	(None, 2, 7, 64)	36928
batch_normalization_13 (Batc	(None, 2, 7, 64)	256
zero_padding2d_27 (ZeroPaddi	(None, 4, 9, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 2, 4, 64)	0
zero_padding2d_28 (ZeroPaddi	(None, 4, 6, 64)	0

Figure 20 - Model Parameter (17-layer CNN) (2)

conv2d_28 (Conv2D)	(None, 2, 4, 64)	36928
zero_padding2d_29 (ZeroPaddi	(None, 4, 6, 64)	0
conv2d_29 (Conv2D)	(None, 2, 4, 64)	36928
batch_normalization_14 (Batc	(None, 2, 4, 64)	256
zero_padding2d_30 (ZeroPaddi	(None, 4, 6, 64)	0
conv2d_30 (Conv2D)	(None, 2, 4, 128)	73856
zero_padding2d_31 (ZeroPaddi	(None, 4, 6, 128)	0
conv2d_31 (Conv2D)	(None, 3, 5, 128)	65664
batch_normalization_15 (Batc	(None, 3, 5, 128)	512
flatten_1 (Flatten)	(None, 1920)	0
dense1 (Dense)	(None, 32)	61472
dropout_2 (Dropout)	(None, 32)	0
dense2 (Dense)	(None, 32)	1056
dropout_3 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 6)	198
reshape_3 (Reshape)	(None, 6)	0

Figure 21 - Model Parameter (17-layer CNN) (3)

This model is a deep CNN architecture. It takes an input shape of (1, 2, 548). Then it goes into two layers of zero padding to get it into the correct shape to be fed into our first convolutional layer. Then we pass it into our first set of convolutional layers that have a filter size of 3x3 and a total of 16 filters per layer. We then add a batch normalization layer in our model to standardize our weights so that they are learned in an effective fashion. We repeat these blocks of convolutional layers with increasing number of filters from 16 to 32 to 64 to 128. We also incorporate L2 kernel regularization for better performance. In all the layers we use 'relu' activation since it performs the best for our type of data. Following this we flatten our output from the convolutional layers and add two fully connected layers of size 32 with the activation function 'relu' and kernel initializer 'he_normal'. We also add dropout of 0.3 and 0.2 after each dense layer. Finally, we have a softmax layer corresponding to our six possible labels.
The loss function we are using is 'categorical_crossentropy' and we are using 'adam' optimizer for optimization. We will calculate train accuracy of our model and validation accuracy.

4.4 Summary:

In this chapter we briefly discussed the methodology we followed regarding our project. We started with data generation with the help of 5G Toolbox available on MATLAB 2020 version. Once we had sufficient data, we opted extracted the PSS1 of each wave such that our data is preprocessed. Finally, once data was prepared, we subjected it to our classification neural network that helped us detect impairment type and level. All these steps were carried on recursively to get the optimum results.

CHAPTER 5: RESULTS

Our results will be presented by the training and validation accuracies of our deep learning models. In this chapter, we will document the results in three phases. A brief description is given as follows.

5.1 CNN – 4 for AWGN

Initially we made a dataset that only consisted of variable noise lying in the range of 0 – 30 dB. The dataset comprised of 300 samples. When this was fed to a CNN with 4 layers, the validation accuracy lied around 63% which represented that the model was underfitting. Thus we needed to enhance the dataset.

Then we extended the dataset to 3000 samples (dataset consisted of PSS-1 of 5G waves only affected by white noise). Once this data was fed to a 4 layered CNN, we had reasonable results. Figure 21 represents the accuracy attained via training our deep learning model on the said data.

ACCURACY	PERCENTAGE
Training Accuracy	94 %
Validation Accuracy	84 %

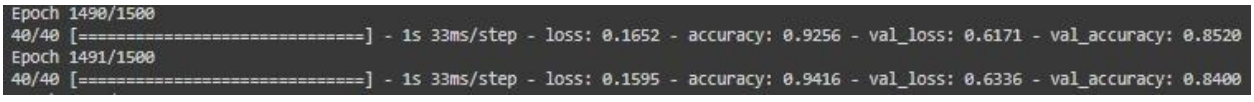

Figure 22 - Phase 1 Accuracy

5.2 CNN – 17 for Multiple Impairments

As the model was performing really well with AWGN as an impairment the next phase, we enhanced the complexity of data by introducing 3 impairments (AWGN, path delays and frequency offset). We made three different datasets, one for each impairment being as variable while the others were kept as maximum (refer to data generation in methodology for clearer insight).

This data was then fed to a 17 – layered CNN and following results were produced. Figure 22 (Phase 2 Accuracy) shows the accuracy when the model was run for 500 epochs.

ACCURACY	PERCENTAGE
Training Accuracy	81%
Test Accuracy	76%

```
Epoch 498/500
40/40 [==============================] - 1s 31ms/step - loss: 0.3924 - accuracy: 0.8133 - val_loss: 0.5506 - val_accuracy: 0.7600
Epoch 499/500
40/40 [==============================] - 1s 31ms/step - loss: 0.4201 - accuracy: 0.8054 - val_loss: 0.5393 - val_accuracy: 0.7620
```

Figure 23 - Phase 2 Accuracy

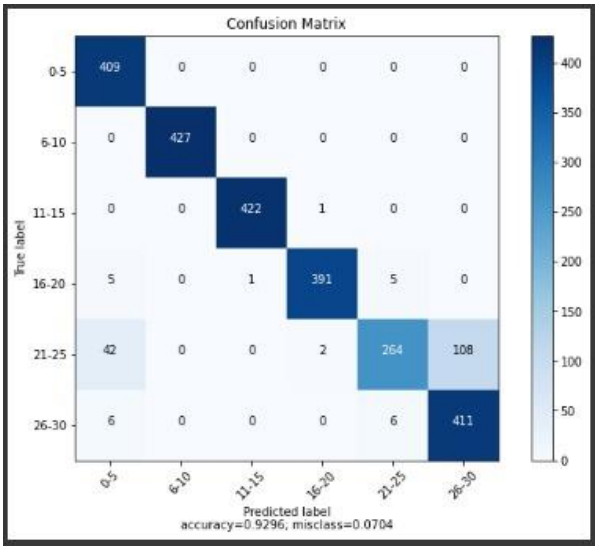

Figure 25 - Phase 2 Confusion Matrix

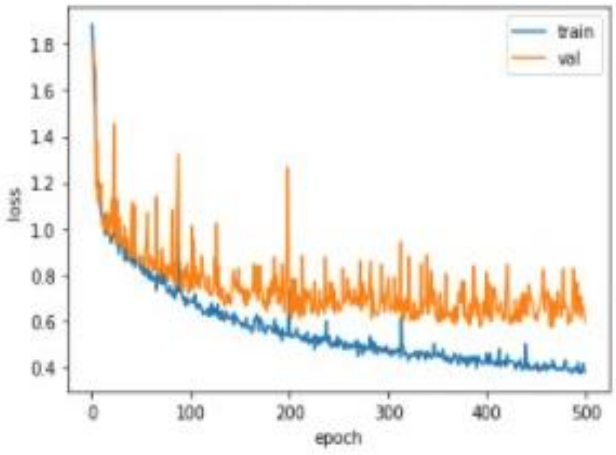

Figure 24 - Phase 2 Loss

Figure 23 represents the confusion matrix which is the illustration of how well our model has predicted correct labels for our test set. Similarly, figure 24 (Phase 2 Loss) illustrates the trajectory of the loss function when plotted.

5.3 CNN – 4 For Multiple Impairments

As the model in phase two wasn't performing up to par even after further increasing the data and fine tuning, we had to change the model architecture. After much considerations, we improved upon our 4 – layered CNN by increasing the number of parameters and by tweaking the hyperparameters, which produced the desired results.

ACCURACY	PERCENTAGE
Training Accuracy	96%
Validation Accuracy	82%

```
- 0s 10ms/step - loss: 0.2150 - accuracy: 0.9684 - val_loss: 0.8267 - val_accuracy: 0.8080
- 0s 10ms/step - loss: 0.2321 - accuracy: 0.9688 - val_loss: 0.8499 - val_accuracy: 0.8140
```

Figure 26 - Phase 3 Accuracy

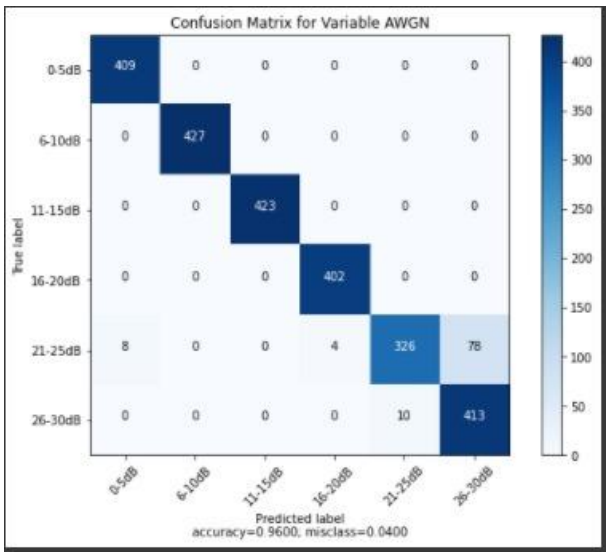

Figure 27 - Phase 3 Confusion Matrix

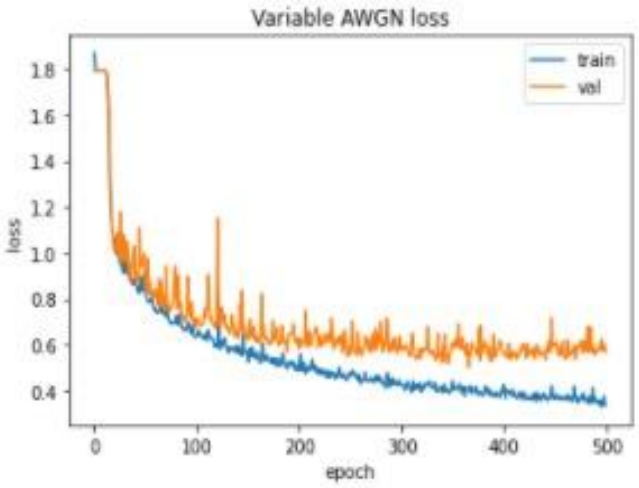

Figure 28 - Phase 3 Loss

Figure 25 shows the working of our model when it was run over with our dataset considering 500 epochs. Once the model had been trained, Figure 27 shows the results of test set in the form of a confusion matrix. Finally, Figure 28 further validates our results by representing the loss function decreasing trend.

5.3 Testing For Multiple Impairments

Finally, we built another notebook to test our impaired 5G Waveforms. We saved the best performing models for all 3 classes i.e CNN-4 and loaded their Tensorflow models onto our notebook. We created a sample wave with the preset impairments of 22dB AWG Noise, 17us of Delay Spread, and 1.4kHz of Frequency Offset. We loaded this data on to a CSV file and uploaded it on our new testing notebook.

After preprocessing and loading the saved models, we used the model.predict() function to test the results on our sample wave. We found that all three models correctly classified the impairment levels and therefore the experiments were a success.

```
predictions_noise = model_noise.predict(final_data)
predictions_noise=(predictions_noise > 0.5)
print(predictions_noise)

list_label=["0-5","6-10","11-15","16-20","21-25","26-30"]
print("Labels for AWG Noise (dB): ",list_label)
```

```
[[False False False False  True False]]
Labels for AWG Noise (dB):  ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30']
```

Figure 29 – Testing for AWG Noise

```
predictions_path_delay = model_path_delay.predict(final_data)
predictions_path_delay=(predictions_path_delay > 0.5)
print(predictions_path_delay)

list_label=["0-5","6-10","11-15","16-20","21-25","26-30"]
print("Labels for Path Delays (us): ", list_label)
```

```
[[False False False  True False False]]
Labels for Path Delays (us):  ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30']
```

Figure 30 – Testing for Path Delays

```
predictions_frequency_offset = model_frequency_offset.predict(final_data)
predictions_frequency_offset=(predictions_frequency_offset > 0.5)
print(predictions_frequency_offset)

list_label=["0-1","1-2","2-3","3-4","4-5","5-6"]
print("Labels for Frequency Offest (kHz): ", list_label)
```

```
[[False  True False False False False]]
Labels for Frequency Offest (kHz):  ['0-1', '1-2', '2-3', '3-4', '4-5', '5-6']
```

Figure 31 – Testing for Frequency Offset

5.4 Summary

Thus, we could summarize our results into three separate phases. Initially, in phase 1, we started with a relatively simpler dataset that has white noise (awgn) as the only impairment. We designed, tested and fine-tuned our deep learning model for this dataset. Once we had acceptable results, we stepped onto phase 2 in which we generated a comparatively complex data set that included a multiple impairment. We opted a new model architecture, that was a 17 layered CNN and generated better results with it. Finally, in the final phase, we reverted back to our initial CNN model but with additional fine tuning in order to improve the results for our dataset at hand. Keep in find that we designed a separate CNN model for each impairment as the model architecture has variance from impairment to impairment. For documentation purposes, only the results of variable AWGN have been presented.

CHAPTER 6: CONCLUSION & FUTURE WORK

Although 5G has been deployed in some countries of the world, yet it is in testing phase not only in Pakistan but also many other countries. According to 3GPP (3rd Generation Partnership Project), 5G is going to last for more than a year while undergoing major improvements and upgrades such that it could provide the data rate, latency and coverage it promises.

To ensure that all 5G requirements are met, thorough research and development is going on to optimize the working and the design of transmitter, receivers and the channels.

Our research project provides a novel approach to determine the channel characteristics by employing the deep learning approach. The characters estimated by our machine learning models are of reasonable credibility owing to the accuracy results we claim. These characters could be further used to design the channel more robustly with better accuracies.

Although we have targeted just a few of these characters (Impairments) as a start, but this work could serve as a basis to design better and complex models (RNNs) that would work for real time data as it would cater all the impairments a 5G OFDM wave could be subjected to.

A few future prospects for our project have been discussed as follows.

6.1 Radio Transformer Models (RTNs)

Although using the convolution neural networks (CNNs), we were able to attain reasonable results when it comes to detect the type and the level of a specific impairment yet the work we have done was with limited number of impairments (AWGN, path delays, frequency offset) yet in real life scenarios, there are many other impairments (such as IQ imbalance, multi path fading, phase offset etc.) a 5G wave could get subjected to. Using a much more complex model will not only cater all the impairments on hand but it will also enhance the accuracies of our results regarding the type and level of impairments.

Radio Transformer Networks (RTNs) could come in handy in this regard. As explained in the literature review, the main idea of such models is to use attention as a way to further improve classification. Attention models focus on a particular segment of the input data and remove variations to make downstream tasks easier and help in reducing their complexity. These features would increase the accuracy of our results by taking into consideration the time series characteristic of 5G OFDM waves.
Thus, the next step should be to make a dataset that would consist all the potential impairments a 5G OFDM wave could be subject to and ultimately use an RTN architecture to process this data.

6.2 Web Based Portal and Application

The next step should be to make to make our work useable for the community, the next future goal is to design a web-based portal that could be used by the research community to test their data and model. Moreover, an application for telecommunication sector could help them use it at receiver end where they could deploy it and it would not only find out the CSI (Channel State Information) but will also use to estimate the channel.

6.2.1 Web Portal

The web portal will be meant for the research community. We plan to design it in such a way that its interactive in nature. Various deep learning models who have reasonable accuracies will be deployed and researches could use them to test their data for impairments that could help them not only detect and validate their results but also our models. They could also suggest modifications to the models that would be a further help to the community as it would enhance the accuracy of the models.

6.2.2 Application Design

The application design will be meant for the industrial community. The application is planned to be installed at the receiver end where we have the incoming data. The user interface design of our application would be such that it would intake the wave data, extract put the PSS and would list down the impairment types and level the wave has. This information would be further used by the application for the process of channels estimation. We would then compare the latency and accuracy of this channel estimation to the currently used pilot-based techniques and would ultimately apply respective modification before it is set for industrial use.

6.3 Summary

In this chapter we presented the concluded project by explaining the impact it could have in the domain of digital communication. After that, we presented some future prospects for our project that included a web portal design to aid the research community and an application that could be deployed by the industry once it has undergone rigorous testing.

References:

1. O'Shea, T.J., Corgan, J. & Clancy, T.C., 2016. Convolutional Radio Modulation Recognition Networks. *Engineering Applications of Neural Networks*, pp.213–226.

2. O'Shea, T.J. et al., 2016. Radio transformer networks: Attention models for learning to synchronize in wireless systems. *2016 50th Asilomar Conference on Signals, Systems and Computers*.

3. O'Shea, T. & Hoydis, J., 2017. An Introduction to Deep Learning for the Physical Layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4), pp.563–575.

4. Farsad, N. & Goldsmith, A., 2017. Detection Algorithms for Communication Systems Using Deep Learning.

5. Gizzini, Abdul Karim & Chafii, Marwa & Nimr, Ahmad & Fettweis, Gerhard. (2020). "Deep Learning Based Channel Estimation Schemes" for IEEE 802.11p Standard. IEEE Access.

6. Yang, Yuwen & Gao, Feifei & Ma, Xiaoli & Zhang, Shun. (2019). "Deep Learning-Based Channel Estimation for Doubly Selective Fading Channels". IEEE Access. 7.

7. Ye, H., Li, G.Y. & Juang, B.-H., 2018. Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems. *IEEE Wireless Communications Letters*, 7(1), pp.114–117.

8. S. Rajendran, W. Meert, D. Giustiniano, V. Lenders and S. Pollin, (2018). "Deep Learning Models for Wireless Signal Classification with Distributed Low-Cost Spectrum Sensors," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 3, pp. 433-445.

9. Clancy, C., Hecker, J., Stuntebeck, E., O'Shea, T (2017).: Applications of machine learning to cognitive radio networks. IEEE Wirel. Commun. **14**(4), 47–52.

10. M. Ibnkahla (2000), "Applications of neural networks to digital communications—A survey", *Elsevier Signal Process.*, vol. 80, no. 7, pp. 1185-1215.

11. Guoru Ding, Yutao Jiao, Jinlong Wang, Yulong Zou, Qihui Wu, Yu-Dong Yao, Lajos Hanzo, (2018) "Spectrum Inference in Cognitive Radio Networks: Algorithms and Applications", *Communications Surveys & Tutorials IEEE*, vol. 20, no. 1, pp. 150-182.

12. Keerthi Suria Kumar Arumugam, Ishaque Ashar Kadampot, Mehrdad Tahmasbi, Shaswat Shah, Matthieu Bloch, Sebastian Pokutta, (2017) "Modulation recognition using side information and hybrid learning", *Dynamic Spectrum Access Networks (DySPAN) 2017 IEEE International Symposium on*, pp. 1-2.

Appendix:

Project Poster: