Modulation Identification

Final Report

T.M. Marshall
18007563

Submitted as partial fulfilment of the requirements of Project EPR402
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2021

Study leader: Prof. W.P. du Plessis

Part 1. Preamble

This report describes work that I did in my final year project, designing and implementing an automatic modulation identification system by applying well-known machine learning algorithms.

Project proposal and technical documentation
This main report contains an unaltered copy of the approved Project Proposal (as Part 2 of the report).

Technical documentation appears in Part 4 (Appendix).

All the code that I developed appears as a separate submission on the AMS.

Project history
This project follows the first iteration by Strydom (2018), but my implementation is completely new. I borrowed ideas from Strydom only pertaining to the continuation of his work and reducing the execution time of my models. This project makes extensive use of existing algorithms on convolutional neural networks. Where other authors' work has been used, it has been cited appropriately, and the rest of the work reported on here is entirely my own.

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was John Marshall.

Language editor signature 2021/11/21 *Date*

Declaration

I, Thomas Michael Marshall understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the Internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will be regarded as plagiarism. If there is any doubt about

the authenticity of my work, I am willing to attend an oral ancillary examination/evaluation about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a verbatim copy of the approved Project Proposal.

T.M. Marshall

2021/11/21

Date

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AMC	automatic modulation classification
AM	amplitude modulation
ANN	artificial neural network
API	application programming interface
Adagrad	adaptive gradient algorithm
Adam	adaptive moment estimation
BW	bandwidth
CNN	convolutional neural network
CPU	central processing unit
CR	cognitive radio
CSIR	council for scientific and industrial research
CUDA	compute unified device architecture
DHCP	dynamic host configuration protocol
DSB	dual sideband
FM	frequency modulation
FSK	frequency shift keying
GPU	graphics processing unit
GSM	global system for mobiles
IP	internet protocol
IQ	in-phase and quadrature
LSTM	long short-term memory
MSK	minimum shift keying
PRNG	pseudo-random number generator
PSK	phase shift keying
QAM	quadrature amplitude modulation
RF	radio-frequency
RMSProp	root mean Square propagation
RNN	recurrent neural network
RRC	root-raised-cosine
ReLU	rectified linear unit
SDR	software defined radio
SGD	stochastic gradient descent
SNR	signal to noise ratio
SSB	single sideband
SSH	secure shell
UML	unified modeling language
USB	universal serial bus

Part 2. Project definition: approved Project Proposal

This section contains the problem identification in the form of the complete approved Project Proposal, unaltered from the final approved version that appears on the AMS.

For use by the Project lecturer	Approved		Revision required	

Feedback

To be completed by the student

PROJECT PROPOSAL 2021				Project no	WdP16	Revision no	0
Title	Surname	Initials	Student no	Study leader (title, initials, surname)			
Mr	**Marshall**	**TM**	18007563	Prof WP du Plessis			
Project title	**Modulation Identification**						

Language editor name	Language editor signature
J. Marshall	
Student declaration I understand what plagiarism is and that I have to complete my project on my own.	**Study leader declaration** This is a clear and unambiguous description of what is required in this project
Student signature	Study leader signature and date
	See attachment

1. Project description

What is your project about? What does your system have to do? What is the problem to be solved?

The proposed project entails the engineering design and implementation of a system that automatically differentiates amongst many different modulation schemes of radio-frequency (RF) signals. These modulation schemes include standard analog modulations and a variety of digital modulation schemes.
The Automatic Modulation Classification (AMC) envisaged must be able to process real-world signals, in absence of specific a-priori signal information at the time of interception. This is also known as Blind automatic modulation classification. The frequency band where the signal is intercepted and no other signals overlap is known.
Modulation Identification occurs principally in intelligent receiver design. It is the intermediary step between signal capture and signal demodulation as well as signal intelligence. It is critical that a system that performs this functionality is in place, such that an advanced signal intelligence system or intelligent receiver functions accurately and efficiently. This system is especially important in many Electronic Warfare (EW) systems. Outside of EW, applications include spectrum regulation by bodies such as ICASA and the FCC, for instance in identifying illegal signals. Blind AMC decreases the required a-priori knowledge of signals for successful interception, demodulation and analysis.

2. Technical challenges in this project

Describe the technical challenges that are *beyond* those encountered up to the end of third year and in other final year modules.

2.1 Primary *design* challenges

- The principal design challenge of this AMC system is the design of the classifier algorithm and selection of the underlying features it relies on and the generation of real-world signal training data. This part is required to bear the brunt of the work and identify modulation schemes correct to a specified overall accuracy. It's success relies on several other complex design elements, making it the principal design challenge as it is related directly to the core functionality of this project.
- The different parts of the system depend on and interact with each other. For instance, the classifier will depend strongly on the individual characteristics and quirks of the receiver. These different parts must harmonise in design.
- The system must function robustly on real-world signals.

2.2 Primary *implementation* challenges

- Integrating the parts into the whole such that the system functions in the real signals. In the presently available literature, it is common to find AMC systems that are developed considering simulations only. This project will develop a system that will process real-world signals.
- Optimising the system such that it functions to the proposed design standard on real-world signals.
- Integrating the RF frontend with the embedded hardware platform. This includes swift and reliable bidirectional communication between the two.
- Most contemporary classifier models and algorithms are computationally expensive. Reducing the execution time of the AMC pipeline on an embedded hardware platform presents a substantial design and implementation challenge.

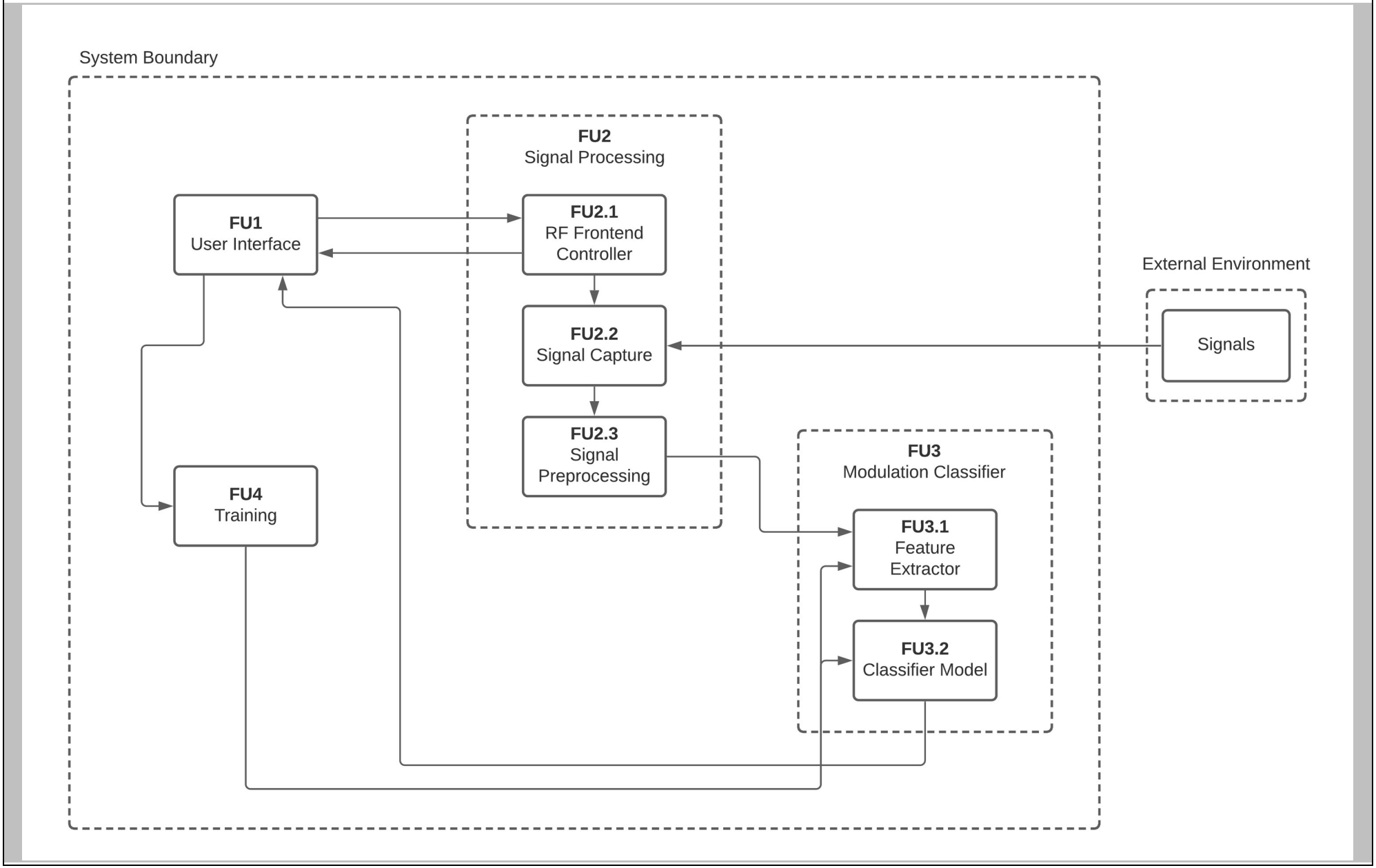
3. Functional analysis

3.1 Functional description

Describe the design in terms of system functions as shown on the functional block diagram in section 3.2. This description should be in narrative format.

- The system's core functional process begins at FU1.1 (functional unit 1.1), the user interface. The user inputs the bandwidth and centre frequency of interest into the system through this interface. The user interface has a normal machine-to-machine variant, but also a human-to-machine variant (in test mode for development and demonstration purposes).
- FU2.1, the RF front-end controller, receives bandwidth and centre frequency designation information from FU1.1. Following FU2.1, in FU2.2 the system then captures incident RF signals from the external environment in the frequency band as informed by the previous step. This process moves on to FU2.3 where the captured signal is filtered and mixed to isolate the true signal of interest. The isolated signal is then preprocessed to maximise the probability of correct classification in FU3.2.
- The predetermined most useful characteristic features of the signal are extracted in FU3.1, and then passed into FU3.2.
- The signal characteristics are passed as inputs to the classifier algorithm to determine the most probable modulation scheme of the isolated signal.
- The result most probable modulation scheme is passed back to the user interface.
- FU4.1 represents the training of the classifier model. This is a crucial function that must be performed at least once prior to enacting the other functions of the system. FU4.1 receives data from the user on which to commence training of the model. This training process proceeds until acceptable accuracy is obtained. The parametric model as determined by training is passed to the classifier function.

3.2 Functional block diagram



4. System requirements and specifications

These are the core requirements of the system or product (the mission-critical requirements) in table format IN ORDER OF IMPORTANCE. Requirement 1 is the most fundamental requirement.

	Requirement 1: the fundamental functional and performance requirement	**Requirement 2**	**Requirement 3**
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	Identify modulation schemes: AM, FM, SSB, LSB, USB and MQAM where M is 8, 16 and 32;MPSK where M is 2, 4, 8; and MFSK where M is 2, 4, 8. Note that 4QAM and 4PSK are indistinguishable.	The system must operate above a minimum accuracy specification for the identification of the prescribed modulation schemes.	The system will operate in an electromagnetic signal environment with a minimum Signal to Noise Ratio (SNR) as specified below.
2. What is the target specification (in *measurable* terms) to be met in order to achieve this requirement?	The system must be able to identify all the required modulation schemes. Specifically in the case of AM and FM where: 0.1 <= FM modulation index <= 5.0, 0.5 <= AM modulation index <= 1.0.	The system will identify the correct modulation scheme 3 times as often as it will identify the scheme incorrectly, a system arithmetic average accuracy of 75% across all listed modulation schemes.	The system will operate upwards from at least 20 dB SNR.
3. Motivation: *how* or *why* will meeting the specification given in point 2 above *solve the problem*? (Motivate the *specific* target specification selected)	This will fulfil the purpose of this project because the abovementioned modulation schemes are commonplace in common radio systems such as link radios, cellular and WiFi communications.	The AMC system is required to be accurate within it's scope of modulations to identify for it to be useful.	The SNR must be at least as much as is generally required in practice to demodulate signals. GSM is designed to operate at long range with low SNR, and requires at least 9 to 10 dB SNR for demodulation.
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	Recorded messages with known modulation schemes will be transmitted over the air at a particular carrier frequency. The result of the system acting on that carrier frequency will then be carried over to the user interface in test mode.	The accuracy specification will be demonstrated using an automated test mode, that will calculate and return the arithmetic average accuracy over the test data.	A RF spectrum monitor will be used to determine the SNR of the signal of interest.
5. Your own design contribution: what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement*.	The training data for the classifier will be generated and recorded in an automated fashion in real-world environments. The classifier algorithm and training algorithm implementation will be done from first principles.	Design, feature selection and implementation for the classifier such that it can operate at the prescribed level and reconciling the classifier model with the quirks of the receiver being used for real-world signals.	This requirement factors in holistically with the others. In particular, care must be taken in the construction of the training set to achieve acceptable performance at relatively low SNR. The signal preprocessing also pertains to this.
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	An off-the-shelf modulator will be used to generate the signals that will be broadcast and received to make up the training set and the test signals.	No off-the-shelf components will be used to realise this requirement.	No off-the-shelf components will be used to realise this requirement.

System requirements and specifications page 2

	Requirement 4	Requirement 5	Requirement 6
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	The system will operate on the FM radio, TV, GSM and ISM frequency bands. There will be no signals overlapping with the desired signal in these bands.	The execution time of the entire AMC pipeline must be reduced such that the potential action can be taken immediately.	
2. What is the target specification (in *measurable* terms) to be met in order to achieve this requirement?	The system will only operate on the frequency range of 90 MHz to 1750 MHz.	The system must produce a result after at most 5s since the directive was given to identify the modulation of a signal acting in a specific frequency band.	
3. Motivation: *how* or *why* will meeting the specification given in point 2 above *solve the problem*? (Motivate the *specific* target specification selected)	The frequency bands specified are those where signals of interest occur, this includes radio and television broadcasts, as well as cellular and WiFi signals.	Reducing the execution time of the system increases the overall utility and usability of the system. It is widely understood that processing radio signals is primarily useful if done quickly.	
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	This requirement will be demonstrated by showing the functionality of the system for signals at the high and the low extreme of operation on the frequency spectrum.	The system will be timed with a integrated automated timing system in test mode from the moment a signal is received to the moment a modulation identification result is presented.	
5. Your own design contribution: what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement*.	The target operating frequency range must be considered in detail during the signal processing phase as well as the compilation of the training set and the selection of features, to ensure the system will function across the entire range.	Several aspects of the classifier algorithm and sub-routines of the system will be optimsed in such a way that they run to completion in less time than they would prior to this optimisation.	
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	An off-the-shelf Software Defined Radio will be used to provide the RF front-end in partial fulfillment to this requirement.	No off-the-shelf components will be used to realise this requirement.	

5. Field conditions

These are the REAL WORLD CONDITIONS under which your project has to work and has to be demonstrated.

	Field condition 1	Field condition 2	Field condition 3
Field condition requirement. In which field conditions does the system have to operate? Indicate the one, two or three most important field conditions.	The system will operate under normal indoor and outdoor conditions for average minimum and maximum temperatures in Pretoria.	The system will operate in a dry environment, protected from moisture.	
Field condition specification. What is the specification (in measurable terms) for this field condition?	The temperature will be in the range from 5°C to 30°C	There will be an absence of liquids, in any form, in contact with the system.	

6. Student tasks

6.1 Design and implementation tasks

List your primary design and implementation tasks in bullet list format (5-10 bullets). These are *not* product requirements, but *your* tasks.

- Control of a SDR based RF frontend capable of filtering, sampling and mixing specific frequency bands in order to isolate a radio signal.
- The creation of a labelled high quality training set of real-world recorded signals assembled with a designed automated system as to train the classifier model with.
- The identification and extraction of useful features to use for AMC in the model.
- The design and training of a classifier model to the satisfactory performance standard.
- Reconciling the hardware and the software; more specifically fitting the classifier model to the imperfections of the SDR receiver and RF frontend that is used.
- Design and implementation of the user interface.
- Optimisation of the AMC pipeline.
- Design and implementation of automated test mode.

6.2 New knowledge to be acquired

Describe what the theoretical foundation to the project is, and which new knowledge you will acquire (beyond that covered in any other undergraduate modules).

This project contains ample topics which have not been covered by this student's university coursework. A brief summary of topics include:
- RF and the background knowledge that goes along with it, including RF signal preprocessing (this includes efficent implementation of variable digital filtering).
- SDR systems.
- Intelligent Systems applied to digital signal processing, and the application of intelligent systems in real-world environments.
- In depth knowledge of analog and digital modulation schemes.
- Optimisation of embedded computer programs to reduce execution time and memory usage.
- Composing useful real-world signal and simulated signal datasets for training classifier models.
- Selecting and optimising features and classifier architectures to best suit the characteristics of the real-world system to best account for the characteristics and quirks of the hardware and design choices.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Engineering, Built Environment and Information Technology
Department of Electrical, Electronic and Computer Engineering

19 May 2021

Approval of Proposal for EPR 402 Project

Dear Prof. Hanekom,

This letter serves to confirm that I have approved the proposal written by Thomas Marshall, student number 18007563, for EPR 402 Project.

Sincerely,

Warren du Plessis
Professor

Room 13-21, Engineering I Building
University of Pretoria, Private Bag X20
Hatfield 0028, South Africa
Tel +27 (0)12 420 2951
Email wduplessis@up.ac.za
www.up.ac.za

Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie
Departement Elektriese, Elektroniese en Rekenaar Ingenieurswese
Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo
Kgoro ya Boetšenere bja Mohlagase, Elektroniki le Khomphutha

Part 3. Main Report

1. Literature study

1.1 Introduction

Automatic modulation classification (AMC) is the process of identifying the radio-frequency (RF) modulation scheme used to produce a signal, in absence of any a-priori signal information, other than the signal itself. The foremost goal of this concept is to bridge the gap between signal capture and signal demodulation. This is a very important function of cognitive radio (CR) systems. AMC is most useful in the context of intelligent receiver design and spectrum dominance for a variety of civilian and military purposes. In the case of the former, this capability can pave the way for the design of intelligent communications systems.

In essence, there are broadly two approaches to AMC: feature-based and likelihood-based [1]. The latter is based on minimizing the probability of false classification and comparing that value against a threshold. This approach, however, suffers from extremely high computational complexity [1]. This project follows the former approach of feature-based detection. In a feature-based approach, a decision is made based on any number of observable values which prove to be indicative of RF modulation schemes.

1.2 Traditional methods

Developing feature-based AMC is a challenging task principally since the features that are indicative of RF modulations are often some combination of elusive, abstract and complex [2]. Examples of such features include various statistically significant cyclic features and higher-order moments and cumulants [3]. These types of features provide good performance on the problem but are difficult to identify and computationally expensive to calculate. The feature-based approach to AMC has opened up the door to machine learning and deep learning approaches to this problem. In a learning-based approach, the features and their relationship to the RF modulation scheme can be inferred by the algorithm.

1.3 Machine learning methods

With the increased access to greater computing over time, the feature-based learning approach has become more common. This approach shows greater promise than likelihood-based approaches for the task of AMC [1]. In addition to being less computationally expensive to execute (once training has been completed), the learning-focused feature-based methods are shown to be more adaptable in fluctuating real-world environments than their feature-based counterparts [3]. In [4] it was shown for the first time that an artificial neural network (ANN) approach can match and outperform traditional approaches in the task of AMC. This result is verified in [5], kick-starting a new machine learning-based paradigm in AMC.

1.4 Machine learning

1.4.1 Artificial neural networks

Machine learning is a term used to describe the study of a class of algorithms that learn by processing examples and figure out how to perform functions without explicit programming through heuristic guidance. In [6], Rosenblatt proposed what would lead directly to the first neural networks. ANNs are comprised of many fully connected nodes, inspired by the biological structure of neurons in animal brains. ANNs are considered the cornerstone of modern machine learning.

1.4.2 Convolutional neural networks

Convolutional neural networks (CNNs) are an extension of ANNs that incorporate the mathematical convolution of the input with filter kernels before feeding into the ANN portion. These two parts are commonly referred to as convolutional layers and the dense layers respectively. CNNs also typically contain pooling layers where values are combined in some way to reduce the dimensionality and hence the computational complexity of the following dense layer. CNNs mark a pivotal moment in the development of deep learning due to their potency in computer vision applications [7].

1.4.3 Activation functions

Neural networks are not off-the-shelf algorithms. There are many design choices involved to design a neural network. The activation function of a neural network represents the fashion in which a neuron (or node) processes its inputs. Many activation functions have been used in academic literature, all of them with different mathematical properties. It is undisputed that the activation function for modern neural networks should be nonlinear as this effectively produces a universal approximator in a multilayer network. Popular activation functions include Sigmoid, Rectified Linear Unit (ReLU) [8], Leaky ReLU and the hyperbolic tangent function. The ReLU activation function is enjoying the spotlight due to its combination of high performance and low computational complexity. The ReLU activation function has been shown to outperform others in multilayer artificial neural networks as well as convolutional neural networks. [9]

Generally, the output layer of a neural network uses the same activation function as all of the previous layers. However, in the case of categorization, it is helpful to transform the output from class scores into a probability distribution to indicate predicted class probabilities. This can be done with the softmax function [10]. This is exceedingly helpful because this allows us to start attributing more depth to the answers given by neural networks.

1.4.4 Loss functions

The goal is to minimize the error of the model. Mathematically this is equivalent to minimizing the loss (also called the cost) function. The loss function of an optimizer gives a metric for the correctness of its estimation, this error metric can then be used for the estimator to learn. The most common loss function is quadratic loss (also called least squares). However, in the case of classification, there are loss functions that are suited to the problem at hand. The cross-entropy function [11] is borrowed from the field of information theory and presents a metric for the difference between two probability distributions. This is a good pairing for the softmax function because it produces a probability distribution.

1.4.5 Training algorithms

Once the error is obtained, the error must be used in some way to update the parameters of the network such that it can learn. Firstly, this error must reach the whole network, not just the output. Nowadays the dominant method of doing this is through the backpropagation algorithm [12]. This algorithm describes how the error should be distributed throughout the network, but it does not stipulate how that error should be used to update the parameters. In principle, this is based upon calculating the partial derivative of the loss with respect to the specific parameter to update.

1.4.6 Optimizers

The function of optimizer algorithms is to describe how the error should be used to update the network parameters. The backpropagation algorithm is almost exclusively used together with the gradient descent algorithm. Gradient descent is a regression algorithm that steps towards the local minimum of a multivariate function, in this case, the neural network. In Stochastic Gradient Descent (SGD) the network parameters are updated by the weighted error, this weight is called the learning rate.

In recent times SGD has generally fallen out of use in favour of better optimizers, such as Root Mean Square propagation (RMSProp), Adaptive Moment Estimation (Adam) [13] and Adaptive Gradient Algorithm (Adagrad) [14]. In the short term (relating to the number of training passes of the dataset) the optimizer algorithm is shown to have a substantial impact. Over the long term, it has been shown they all perform similarly.

1.4.7 Batch training

There are two ways to process the training data. Firstly, updating the parameters after N samples. Secondly, updating the parameters after passing over the entire dataset. These approaches are referred to as mini-batch and batch respectively. Batch gradient descent has a high propensity of moving towards a saddle point. A saddle point is somewhere in the

multivariate function where the gradient will loop around and never allow the solution to progress. Hence mini-batch gradient descent is preferred. This method has been shown to speed up convergence relative to other methods.

1.4.8 Regularization

The distinction between statistical inference and machine learning lies therein that the former seeks to infer facts about a population via a sample. The latter seeks to find generalizable predictive patterns. Hence the importance of regularization, a method used to combat overfitting the training dataset, encourage generalizability and simpler models.

There are many ways to accomplish this goal, in the literature Lasso Regression (L1) and Ridge Regression (L2) are commonly used [15]. These two methods penalize the loss function for absolute magnitude and square magnitude of network parameters respectively. The effect of confining the domains of the parameters is decreased overfitting and increased generalizability. In recent publications, a new regularization method has emerged which outperforms traditional regularization in many applications.

Dropout works by training a randomly sampled sub-network of the whole neural network for every backpropagation [16]. In essence, one ends up with an ensemble of smaller networks that consolidate their intuition in the final model. This has been shown to perform very well.

1.4.9 Datasets

In recent years datasets for RF machine learning have begun to emerge. In the case of modulation identification there are the RadioML2016 and more comprehensive RadioML2018 datasets, which consist of many modulated signals (both real and synthetic) across various receivers, signal to noise ratios (SNRs) and other parameters. At the time of writing, RadioML2018 is the most popular and comprehensive dataset for modulation identification and is often cited and used as a benchmark in papers presenting new work on AMC. However, some authors do develop their own datasets from first principles [17]; similarly in this report the dataset for AMC was created from first principles.

1.5 AMC using machine learning

The CNN approach [18] shows promise in this category and far outperforms traditional feature-based approaches. This is because this type of structure has two important characteristics: firstly it is shift-invariant and secondly, it can learn matched filters for the application at hand.

In [19] the author shows that the internal structure of the CNN has a significant bearing on the quality of the solution that is reached. The author finds that a CNN with two sets of two convolutional layers and a single max-pooling layer offers the best performance in this application. In [17] the authors propose several CNN-based models. The model that performs the best consists of two convolutional layers and no pooling layers.

The recurrent neural network (RNN) is a structure that was created to be well suited for temporal environments being time-invariant, whereas the CNN is for spatial environments being shift-invariant. Making use of a sequence of time samples to determine RF modulation is intuitively a temporal activity rather than a spatial activity. Despite this authors generally do not consider the RNN for this application. It has however been shown to be very effective [20]. The reason for this disregard is thought to be because RNNs are notoriously difficult to train properly, take too much time to train and suffer from high computational complexity.

Hence there is much value in a simpler solution that offers similar performance. However, there is no doubt that the future of cutting-edge AMC lies in the hybridization of structures, such as combinations of CNN and RNN elements [21].

1.6 Modulation types

Modulation is the process of encoding information such that it may propagate as a wave on a transmission line at a specified carrier frequency. There are two types of modulations: analogue and digital, so named because of the type of information they encode. The most ubiquitous analogue modulation schemes are amplitude modulation (AM) and frequency modulation (FM). Commonly used digital modulation schemes include quadrature amplitude modulation (QAM), phase-shift keying (PSK) and frequency-shift keying (FSK).

The fundamental definition of modulation is presented in [22] where $A(t)$ is the amplitude, ω_c is the carrier frequency and $\phi(t)$ is the phase shift:

$$s(t) = A(t)\cos[2\pi f_c t + \phi(t)] \tag{1}$$

Varying $A(t)$ results in AM, varying f_c produces FM and FSK. Subsequently varying the phase $\phi(t)$ over time gives phase modulation, the foundation for PSK. QAM is delivered through a combination of varying amplitude and phase. From this equation, the mathematical models for all of the above modulations can be derived, since they are all some combination of varying the mentioned three parameters either continuously or discretely.

1.6.1 Amplitude modulation

In AM the amplitude of the carrier signal is varied to represent the message signal. There are many different kinds of AM signals, in this case, the interest lies in double-sideband (DSB) and single-sideband suppressed carrier (SSB-SC).

From [22] the equation for AMDSB is given by

$$s(t) = \left(1 + \frac{m(t)}{A}\right)\cos(2\pi f_c t) \tag{2}$$

Where $m(t)$ is the amplitude of the message signal, A is the amplitude of the carrier signal and the modulation index is represented by the factor $1 + m(t)/A$. This process results in three spectral features: two information carrying sidebands symmetric around the carrier frequency, and the carrier frequency itself. AMSSB-SC has two forms: upper sideband (USB) and lower sideband (LSB). These variations can be achieved by filtering out either the upper or lower

side of the AMDSB spectrum.

1.6.2 Frequency modulation

The mathematical description of frequency modulation is given by

$$s(t) = A_c \cos\left(2\pi f_c t + \frac{f_\Delta}{f_m}\sin(2\pi f_m t)\right) \tag{3}$$

A_c is the carrier amplitude, f_c is the carrier frequency, f_Δ is the peak deviation from the carrier frequency and f_m is the frequency of the sinusoid approximating the baseband signal. The modulation index for FM is given by the factor f_Δ / f_m.

1.6.3 Frequency shift keying

The former modulation schemes have encoded continuous-time analogue waveforms. The subsequent modulation schemes all encode digital information represented by bits. Digital modulation schemes represent bits through symbols. A symbol is comprised of some combination of bits and appears for the discrete-time demarcation T, the symbol period. FSK is the discrete-time digital counterpart of FM, in its basic form it offers two distinct frequencies to represent the symbols for 1 and 0. These two symbols are given by

$$\begin{aligned} s_0(t) &= A_c \cos(2\pi(f_c - f_\Delta)t) \\ s_1(t) &= A_c \cos(2\pi(f_c + f_\Delta)t) \end{aligned} \tag{4}$$

Where A_c is the carrier amplitude, f_c is the carrier frequency and f_Δ is the deviation from the carrier frequency. Minimum Shift Keying (MSK) is a subset of FSK where f_Δ = Bitrate/4.

1.6.4 Phase shift keying

In the PSK modulation [23], the bit to symbol mapping is performed by varying the phase of the modulated signal. This modulation introduces constellation maps as seen in Figure 1, which represent bit to symbol mapping for PSK and QAM type digital modulation schemes.

In the PSK modulation scheme, the total available phase of 2π is subdivided into M equal parts, where each arm represents a distinct symbol. The mathematical representation of this concept is given by

$$s_i(t) = \sqrt{\frac{2E}{T}}\cos\left(2\pi f_c t + \frac{2\pi}{M} i\right) \quad \text{for } i = 0, 1 \ldots M \tag{5}$$

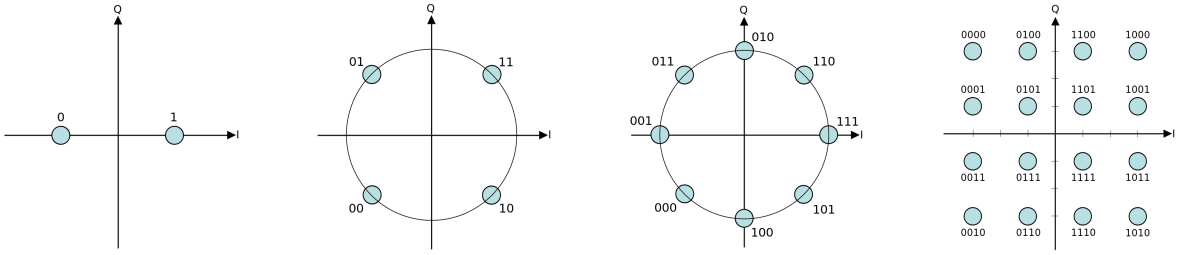
Where M is the order of the constellation and E is the signal energy per symbol.

1.6.5 Quadrature amplitude modulation

QAM works similarly to PSK, but in addition to varying phase, it also varies amplitude, unlocking larger constellations and more bits per symbol. The mathematical representation of QAM is given in [23]:

$$s_i(t) = \sqrt{\frac{2E}{T}}(a_i \cos(2\pi f_c t) - b_i \sin(2\pi f_c t)) \quad \text{for } i = 0, 1 \ldots M \tag{6}$$

Where a_i is the amplitude of the quadrature carrier and b_i is the amplitude of the in-phase carrier. Both the modulation schemes PSK and amplitude shift keying (ASK) can be considered special cases of QAM, where $a_i = 0$ and $b_i = 0$ respectively.



2PSK **4PSK** **8PSK** **16QAM**

Figure 1.
Modulation constellations by Splash under GNU Free Documentation License

1.7 Application of literature to this project

The state-of-the-art in AMC has shifted towards feature-based approaches, more specifically, machine learning approaches. It has been shown that the use of neural networks in AMC delivers high performance and is a potent platform for addressing this problem. Visual convolutional neural networks adapted for this task have been very successful.

The goal of this project was to design and implement a low-cost AMC system that uses modern machine learning techniques to achieve its performance specifications. A convolutional neural network was chosen because of its cardinal function of learning useful filters and property of shift-invariance. This CNN was implemented in Python and also makes extensive use of Graphics Processing Unit (GPU) acceleration through Compute Unified Device Architecture (CUDA) to reduce execution time.

The design of this CNN uses two convolutional layers with cross-entropy loss and a softmax function on the output layer. This design also employs the Adam optimizer to converge on better solutions in less training time. The ReLU activation function was used in this design; ReLU has been shown to outperform other activation functions, particularly, because it is

effective against the vanishing gradients problem. Dropout was used to regularize the model during training, because it has been shown to greatly increase model generalizability in all applications.

In the domain of machine learning, the dataset is arguably more important than the particulars of the model. True to the premise of this project, only real recorded signal data was used to train the model. No simulated data was used. This is significant as the literature study did not encounter any examples where models were trained and operated in a real signal environment. The dataset of recorded modulations was generated using the GNU Radio Application Programming Interface (API), transmitted via a HACK-RF Software Defined Radio (SDR) and received with a low-cost RTL-SDR.

2. Approach

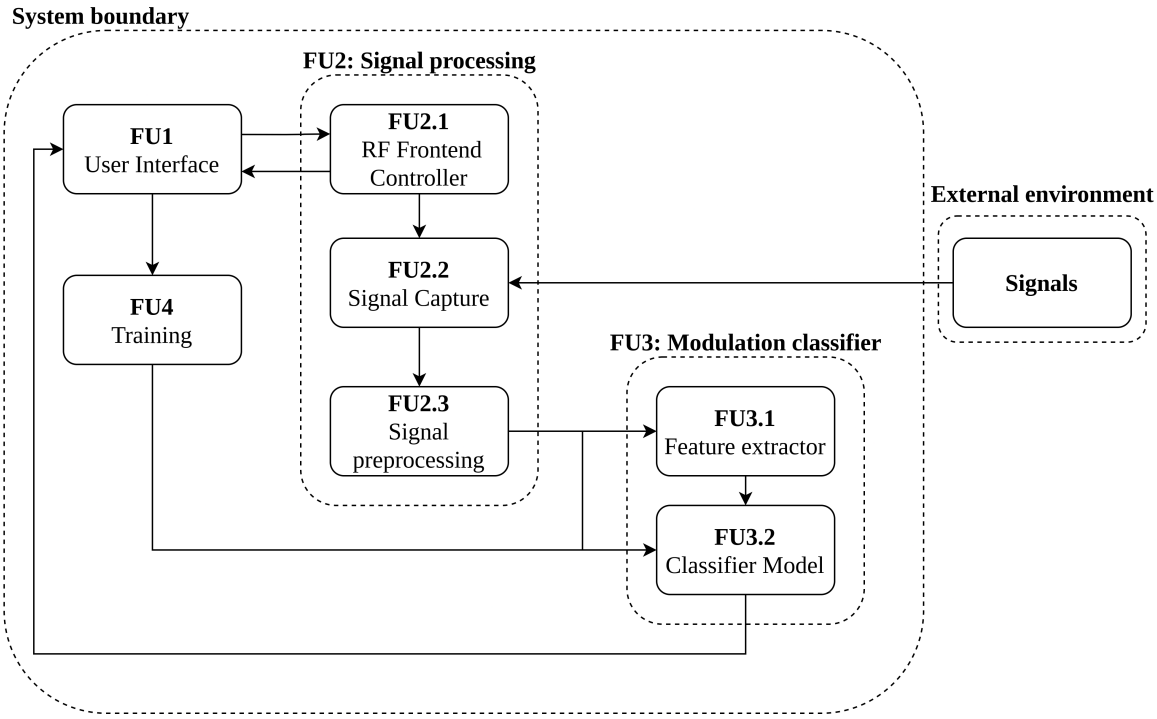


Figure 2.
System functional block diagram

2.1 Design alternatives

There are many design alternatives to consider during the high-level conceptual analysis of an AMC system. Figure 2 presents the functional block diagram of the system.

For the system to be able to receive signals as in FU 2, a RF receiver must be used. AMC and CR systems tend strongly toward using SDR platforms for their RF front end. There are quite a number of SDRs available on the market with greatly varying characteristics. In this work, two SDRs were required, one for transmitting and one for receiving signals. SDRs also perform filtering and decimation and produce a baseband signal. This project requires that the receiving radio is low-cost and able to receive in the range of 90 MHz to 1750 MHz. Table 1 presents the notable characteristics of a number of SDRs that were available for this project.

The baseband signal arrives at FU 3.1 to the feature extractor. There were two ways this could have been approached: use expert-selected features or, feature learning. Together with this, in FU 3.2 there were many considerations for the classifier algorithm, which is deeply intertwined with the feature extractor. Possible classifier algorithms considered were ANNs, deep neural networks (DNNs), CNNs and RNNs. These alternate approaches were evaluated by way of training and testing prototype classifier models using the TensorFlow API in Python.

	RTL-SDR	HACK-RF One	Pluto SDR	Ettus B210
Cost	23.95 USD	319.95 USD	248.95 USD	1472 USD
Operating frequency	25 MHz – 1.76 GHz	1 MHz – 6 GHz	325 MHz – 3.8 GHz	70 MHz – 6 GHz
Maximum Bandwidth	2.4 MHz	20 MHz	20 MHz	56 MHz
ADC/DAC Resolution	8 bits	8 bits	12 bits	12 bits
Receive	Yes	Yes	Yes	Yes
Transmit	No	Yes	Yes	Yes

Table 1.
Comparison of available SDRs

In addition, different structures for the above models, were also evaluated to ultimately select the most appropriate architecture for the AMC system.

Training large machine learning models (FU 4) is known to be computationally expensive, but there are solutions to speed up this process. During this project, the author was kindly granted access to the Council for Scientific and Industrial Research's (CSIR's) Lengau cluster. This opportunity allowed for the evaluation of the merits of deep learning acceleration by large scale multiprocessing as well as GPU acceleration.

	Jetson Nano Dev Kit v3	ODROID-XU4	Raspberry Pi 4B	ODROID-N2+
Cost	99 USD	59 USD	35 USD	83 USD
Clock speed	1.43 GHz	2 GHz	1.5 GHz	2.4 GHz
CPU cores	4	8	4	4
RAM	4 GB	2 GB	2 GB	4 GB
GPU	Yes	No	No	No
Available	Yes	Yes	No	No

Table 2.
Comparison of embedded processing platforms

The embedded processing platform for the final system to run on must be selected based on its ability to run the model and make the specification for execution time. In deep learning, it has become the standard to rely on GPU acceleration, because neural network structures can be implemented largely as a series of vector and matrix operations. This kind of operation is done very quickly and efficiently by GPUs.

2.2 Preferred solution

In this system, there is a requirement for a transmitting SDR and a receiving SDR. The RTL-SDR has been chosen as the receiving SDR, because it is the lowest cost SDR and because it is sufficient in terms of the operating frequency range. The HACK-RF One was chosen to be the transmitting radio. This is because the HACK-RF One can transmit over the full required frequency range and because it works well with the choice of software package. GNU Radio was chosen to operate the SDRs due to its open-source qualities and strong relationship with the HACK-RF One SDR.

The dataset for AMC was generated using the developed data generator script which uses scripts produced with the GNU Radio Companion, by interleaving transmission on the HACK-RF One and reception on the RTL-SDR. The two radios were connected using a short RF coaxial cable. The TensorFlow library was used to prototype and evaluate various models and structures for suitability to this problem. ANNs, DNNs, RNNs and CNNs were considered. Ultimately a CNN structure with two convolutional layers and one dense layer was chosen. This is because it exhibited the best performance in the prototyping tests due to the learning of filters over both channels I and Q. This implementation was done with vector and matrix operations in mind to increase efficiency.

The embedded processing platform was chosen principally based on its suitability to run a deep learning model. The Jetson Nano Dev Kit v3 was chosen for the operation of the system, because of its onboard 128 core GPU. This is ideal for running neural networks that have been formulated as vector and matrix operations.

3. Design and implementation

3.1 Design summary

Deliverable or task	Implementation	Status
The implementation of an interface with a chosen SDR platform to receive and transmit signals.	The student completed this task by selecting the RTL-SDR as well as the HACK-RF for this project and interfacing them with GNU Radio API and the RTL-SDR drivers in Python respectively.	Completed. Section 3.4.1
The generation of a comprehensive dataset in an automated fashion in real-world environments to train the models.	This task was completed by the student by using the GNU Radio API, producing a Python script that automatically generates and transmits signals via the HACK-RF One over a coaxial RF cable to be recorded on the RTL-SDR. Finally, this recorded signal was saved to a data file with its correct label.	Completed. Section 3.4.1
The identification of useful features and development of feature extraction algorithms.	This task was completed by the student by the design and implementation of a CNN, which learns how to perform feature extraction optimally for this problem.	Completed. Section 3.4.2
The design of the classifier model.	The student-designed the classifier model by studying the pertinent literature and developing various prototypes using TensorFlow that were trained and evaluated using the custom made dataset.	Completed. Sections 3.3 and 1.5
The implementation of the classifier model and training algorithm.	The student completed this task by implementing a CNN from first principles in Python. This first principles implementation was also trained.	Completed. Section 3.4.2

The optimization of neural network hyperparameters and other system parameters.	This student completed this task by running tests to determine what the optimal values are for the learning rate, the dropout rate and the number of measurements to make up a result.	Completed. Section 3.4.5
Reduction of execution time during training and operation of the model	The student applied various coding techniques to increase the efficiency and execution speed of the code, as well as employing multiprocessing and GPU acceleration through CUDA to speed up the program.	Completed. Section 3.4.5
Implementation of a user interface for the AMC system.	The student completed this task by creating a command-line based interface for the AMC system.	Completed. Section 3.5.9

Table 3.
Design summary

3.2 Theoretical analysis

3.2.1 Artificial neural networks

An ANN is a nonlinear optimization algorithm that aims to converge on a minimum in a multidimensional problem space. ANNs are made up of many interconnected neurons. These neurons take in several inputs and produce an output; how a neuron does this transformation is called the activation function $g(x)$. The structure of a neuron is given in figure 3.

The mathematical relationship of the input of a single neuron to the output [24] is given by

$$a_j = g(\sum_{i=0}^{n} w_{i,j} \cdot a_i + b_j) \tag{7}$$

Where $w_{i,j}$ is the input weights, b_j the neuron bias and a_i the inputs to the neuron. Crucially this can also be expressed as a simple dot product

$$a_j = g(\mathbf{w_j} \cdot \mathbf{a} + \mathbf{b_j}) \tag{8}$$

ANNs consist of many neurons arranged in columns called layers. These layers are connected by multiplicative and additive parameters called weights and biases. It is called a fully connected network if all of the neurons in the present layer are connected to all of the neurons in the next layer. This combination of neurons is depicted by figure 4. Given this structure of many neurons, it is possible to vectorize the mathematical model for an entire layer:

$$\mathbf{a}^{l+1} = g(\mathbf{w}^l \times \mathbf{a}^l + \mathbf{b}^l) \tag{9}$$

This abstraction is helpful for implementation because it expresses the feedforward of a multilayer ANN in terms of vector and matrix operations, the likes of which modern computers are optimized to perform efficiently.

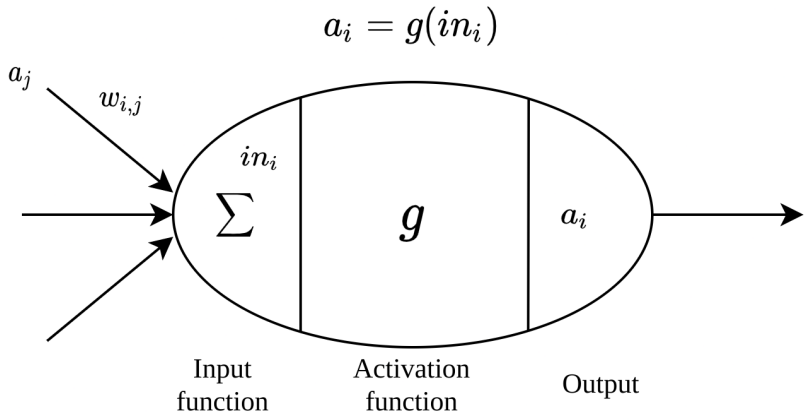


Figure 3.
Artificial neuron structure [24]

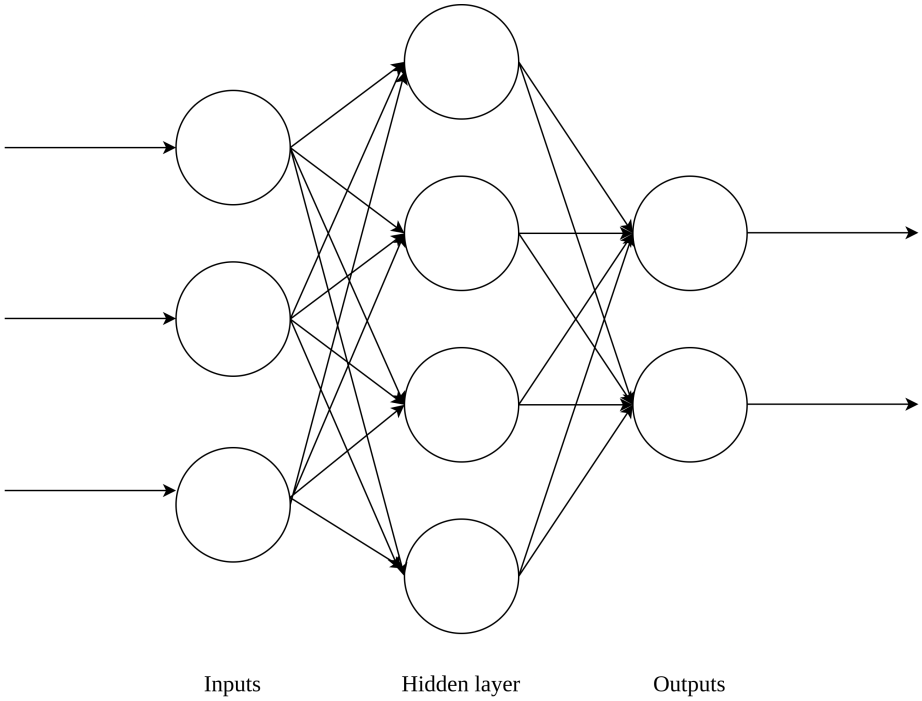


Figure 4.
Artificial neural network structure

3.2.2 Activation functions

Activation functions are a necessary component of ANNs because they introduce nonlinearity into the system, this is part of what gives neural networks their potency. ReLU is a powerful nonlinear activation function that greatly reduces the problem of vanishing gradients. In multilayer networks, the gradient has a propensity to tend towards zero as it is propagated further back into the network, this is called the vanishing gradients problem. The ReLU function is illustrated in figure 5. It is important for later to also define the derivative of the ReLU activation function as it is required in the backpropagation algorithm. ReLU and its derivative are described below:

$$g(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases} \tag{10}$$

$$g'(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{otherwise.} \end{cases} \tag{11}$$

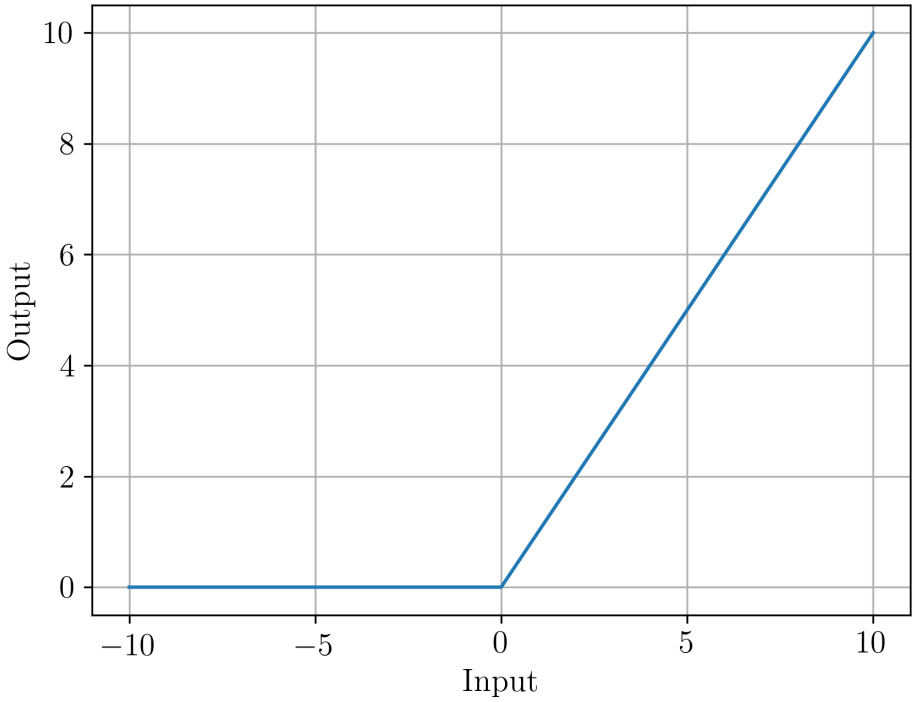


Figure 5.
ReLU activation function

A different activation function was used for the output of the system so that it outputs class probabilities. The softmax function was used on the output because it offers a probability distribution over the categories the system considers. The softmax function [24] generalizes the logistic function to many dimensions and is mathematically defined as

$$\sigma(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \quad \text{for } i = 1, \ldots, N \tag{12}$$

Where $\mathbf{z}$ is the input vector to the softmax function and N is the number of categories of the output.

3.2.3 Categorical cross-entropy loss

To be able to train the structure, it is required to first calculate the error. In this case, it was decided to use the cross-entropy function because it fits well with the softmax function. The cross-entropy function calculates the difference between two probability distributions and is given by

$$L = -\sum_{i=0}^{N} y_i \cdot ln(\hat{y}_i) \tag{13}$$

Since the true label vector is one-hot encoded, the cross-entropy loss formula reduces to

$$L = -ln(\hat{y_c}) \tag{14}$$

Where $\hat{y_c}$ is the estimated probability of the correct class.

The derivative of the cross-entropy loss function is also of interest. Additionally, the cross-entropy derivative simplifies enormously when used in tandem with the softmax function. The cross-entropy derivative is given by

$$\frac{\delta L}{\delta \mathbf{z}} = \hat{y}_i - y_i \tag{15}$$

Where $\hat{y}_i$ is the output of the softmax function and y_i is the one-hot encoded true label vector.

3.2.4 Backpropagation of errors

Now that the starting gradient has been determined, to minimize the loss function this must be related to the weights and biases of the neural network structure. This can be accomplished through backpropagation with gradient descent [24]. The working principle of the backpropagation of errors is to find the partial derivative of the loss with respect to the neural network parameters. To find the loss with respect to any parameter in the neural network the chain rule is used:

$$\frac{\delta L}{\delta \mathbf{w}^l} = \mathbf{z}^l \cdot \frac{\delta L}{\delta \mathbf{z}} \tag{16}$$

$$\frac{\delta L}{\delta \mathbf{b}^l} = \frac{\delta L}{\delta \mathbf{z}} \tag{17}$$

$$\frac{\delta L}{\delta \mathbf{a}^{l-1}} = \mathbf{w}^l \times \frac{\delta L}{\delta \mathbf{z}} \tag{18}$$

For the gradient $\delta L/\delta \mathbf{a}^{l-1}$ to reach the previous layer, the activation function must be taken into account since the output of a layer consists of activated values. This is where the ReLU derivative is used. The ReLU derivative is given by

$$\frac{\delta L}{\delta \mathbf{z}^{l-1}} = \frac{\delta L}{\delta \mathbf{a}^{l-1}} \circ g'(\mathbf{z}^{l-1}) \tag{19}$$

This process can be extended progressively back into the network, hence the name back-propagation of errors. Since the gradient have been found, gradient descent can be applied.

3.2.5 Gradient descent

The gradient descent algorithm is an iterative optimization algorithm for finding the local minimum of a differentiable function. This is applied to train neural networks since a neural network is nothing more than a differentiable multivariate function with a large number of parameters. The general form of gradient descent is given as

$$\hat{\theta}^{i+1} = \hat{\theta}^i - \eta \cdot \Delta\theta^{i+1} \tag{20}$$

Where η is the learning rate, $\hat{\theta}^{i+1}$ is the next estimate of the parameter being optimized and $\Delta\theta^{i+1}$ is the gradient of the loss function with respect to the parameter being estimated.

In the gradient descent method, parameters are only updated once deltas for all of the samples have been calculated. This is not efficient and does not perform well, which is why stochastic gradient descent (SGD) was introduced. In SGD the parameters are updated after every sample. This allows for accelerated convergence in comparison to gradient descent.

3.2.6 Batch modes

The number of deltas aggregated before the parameters are updated can vary anywhere from one (as in SGD) to N (as in gradient descent). The number of deltas that are collected before updating the network parameters is referred to as the batch size and is used for mini-batch mode gradient descent. This batch size can be configured regardless of the optimizer algorithm being used.

3.2.7 ANN training algorithm

Practically the training of neural networks is accomplished by executing the training algorithm for each sample in the dataset. One training pass over of the dataset is called an epoch. Typically, many epochs are required for a model to fully learn the patterns offered by a dataset.

The training algorithm for an ANN is summarized in the following steps:

1. **Feedforward** with some input,
2. **Calculate** the output error,
3. **Backpropagate** the gradient through layers,
4. **Update** the network parameters using the local gradients and the optimizer update rule.

3.2.8 Optimizers

SGD is the baseline optimizer for ANNs. An optimizer algorithm describes how the parameters should be updated using the gradients. In SGD, the parameters are updated by a fixed proportion γ of the local gradients. SGD is considered to be the default starting point, however, some alternatives serve to facilitate faster and better convergence.

RMSProp is an optimizer algorithm for which the learning rate is individualized on a per parameter basis. This method uses a running average of recent gradients for a particular parameter to decide its next learning rate. This gives the successive changes for any particular parameter a sort of momentum. The update rule for RMSProp is given by

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)\cdot\Delta\theta^2 \tag{21}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t}} \cdot \Delta\theta \tag{22}$$

The recommended value for the moving average parameter γ is 0.9. The learning rate η is still configured as normal.

The Adaptive Moment Estimation (Adam) algorithm builds on RMSProp and computes adaptive learning rates on a per parameter basis. The distinction between Adam and RMSProp is that Adam also keeps an exponentially decaying average of past gradients in addition to past squared gradients. This algorithm is considered to be state-of-the-art in machine learning optimization. The update rule for Adam is given as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\Delta\theta \tag{23}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\Delta\theta^2 \tag{24}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{25}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{26}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t \tag{27}$$

The authors of Adam suggest values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

3.2.9 Regularization

Regularization serves to ensure the patterns learned by the neural network are general to the problem and not specific to the dataset. The goal is to implement strategies to increase the generalizability of the solutions, often this means trading off convergence time.

Dropout is the state-of-the-art method for regularization in neural networks that have shown to be very effective in all applications. In dropout [16], a random number of neurons in each layer are retained with probability p, the rest are set to zero. A subnetwork is sampled every time a parameter update has been performed.

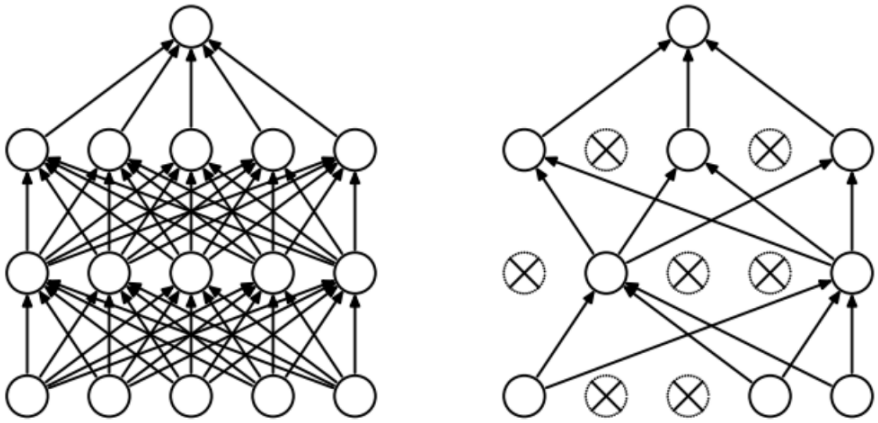


Figure 6.
Illustration of dropout from the original paper [16]

Although dropout is a very powerful method, it should not be applied to the filter kernels in convolutional layers. This has been shown to degrade performance across the board. Hence dropout was only applied to dense layers, excluding the output layer. An important point about dropout must be mentioned: during training time it is sufficient to switch off neurons, bearing in mind that the weights and biases of the deactivated neuron should not be updated. During test time, after training has been completed, it is required that all of the weights and biases in the network be scaled by a factor of p.

3.2.10 Model validation

While a model is training the performance must be quantified. This is done to provide a live metric of how well the training is progressing. In this case, it is typical to use a validation dataset, which is a dataset that the neural network was not trained on and has not seen before. The performance of the model on the validation dataset is then an indication of overall performance, as well as generalizability when compared to the training performance.

In a case like this AMC system, the validation accuracy is of far greater concern than the training accuracy, because the aim is to find highly generalizable patterns for identifying modulation schemes. Keeping track of the validation accuracy during training allows for techniques such as early stopping, where the training is discontinued at the peak of the validation accuracy rather than at the peak of the training accuracy.

3.2.11 Convolutional neural networks

A convolutional neural network is an extension of artificial neural networks that borrows the concept of convolving the input with a filter from digital signal processing. In a CNN, the convolution is generalized for many dimensions, in this case, two and three-dimensional

filters were used. The mathematical notation for convolution is given as

$$h = f \circledast g \tag{28}$$

Where f is the input and g is the filter kernel.

In terms of CNN convolution, some parameters must be looked at. Kernel size represents the dimensions of the filter that is shifted across the input to compute the convolution, this shifting is referred to as stride. In this implementation the stride will be equal to one, meaning the filter shifts by one unit at a time. The convolution of a 2×8 input with a 1×3 filter kernel is shown in figure 7.

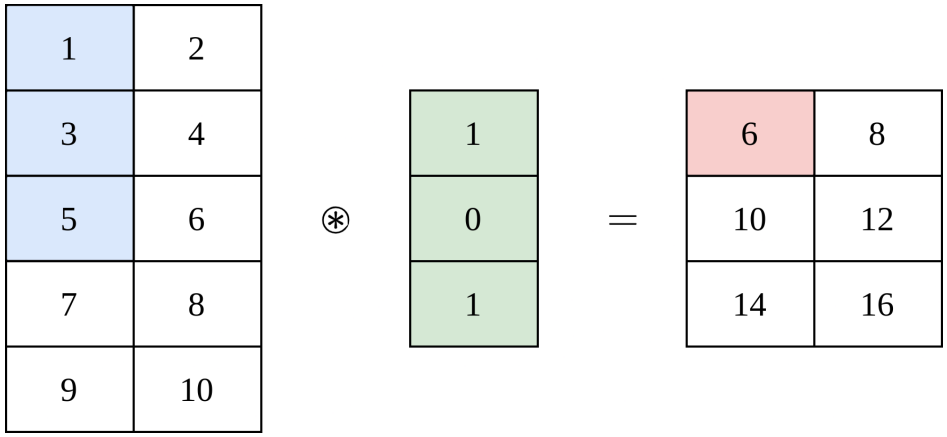


Figure 7.
Illustration of convolution: 2×5 input convolved with 1×3 filter kernel

In a convolutional layer, there are many filters. The aforementioned operation can be stacked for an entire layer as shown in the figure 8. In this formulation, the convolution operations are performed on the same input, but with different filter kernels N times, where N is the number of filters in the convolutional layer.

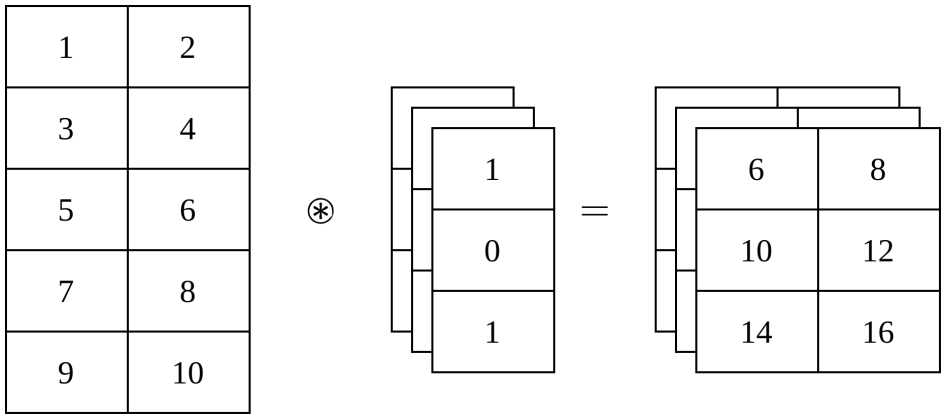


Figure 8.
Illustration of convolution: 2×5 input convolved with many 1×3 filter kernels

Activation functions are also often utilized in the convolutional layer. After the filtering operation, the resultant feature maps are passed through an activation function. In this case, ReLU was applied after filtering.

The utility of CNNs lies in that these filter kernels in the convolutional layer are trainable parameters. Therefore backpropagation of errors can be extended to convolutional layers to train all of the filter kernels. Firstly the author would like to define the forward pass of a convolutional layer as

$$\text{output}(i,j) = \sum_{x=0}\sum_{y=0} \text{input}(i+x, j+y) \circledast \text{filter}(x,y) \tag{29}$$

Then the backpropagation equations are given by the partial derivatives. Take note that when ReLU is applied, then ReLU's derivative must also be applied in the backpropagation step.

$$\frac{\delta L}{\delta \text{filter}(x,y)} = \sum_{x=0}\sum_{y=0} \text{input}(i+x, j+y) \circledast \frac{\delta L}{\delta \text{output}(i,j)} \tag{30}$$

$$\frac{\delta L}{\delta \text{input}(x,y)} = \sum_{x=0}\sum_{y=0} \text{filter}(i+x, j+y) \circledast \frac{\delta L}{\delta \text{output}(i,j)} \tag{31}$$

3.2.12 Im2col

In practice performing convolution is extremely slow, this is because it requires many nested loops to iterate through all of the filters and inputs. This is why the Im2col algorithm was developed, to rearrange convolution in such a way that it can be computed via matrix multiplication. The term image matrix refers to the input of Im2col and the term column matrix refers to the output of Im2col. The performance gains from using Im2col are greater the more filters are used. Im2col also opens up the vectorization of the convolutional layers, and so those computations can then also be calculated on a GPU similarly to the ANN's dense layers.

The steps for convolution via Im2col are summarized:

1. Flatten all filter kernels and let each flattened kernel be a row in a filter matrix,
2. Flatten each convolution window of the input as per the filter kernel size and arrange each flattened window as the column in an input matrix,
3. Perform matrix multiplication of the input matrix by the filter matrix.

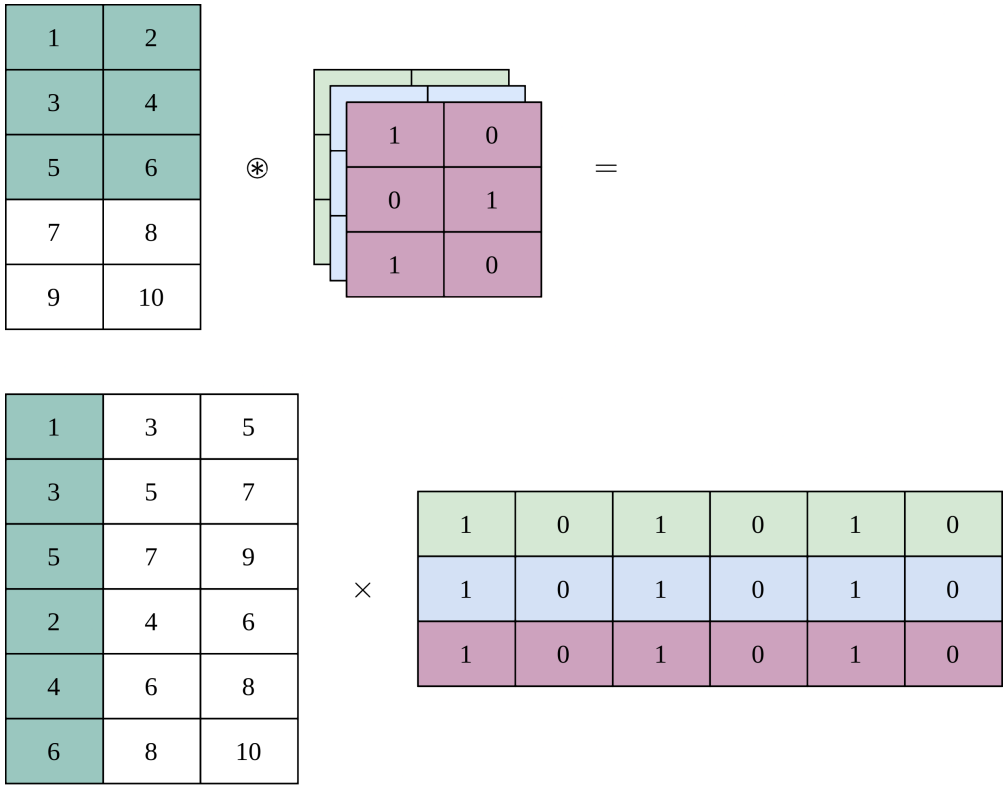


Figure 9.
Illustration of convolution using Im2col

In the implementation of the forward pass of the convolutional layer, it is not necessary to compute the inverse of Im2col such that the result is transformed back into image form. However, during the backpropagation of the convolutional layer, it is necessary to have this ability. The Col2im algorithm reverses Im2col.

The steps for performing Col2im are summarized:

1. Take a column *n* from the column matrix,
2. Reshape the column to filter kernel dimensions,
3. Add this filter kernel to the subset of the image matrix where the *n*th column would have been taken from in Im2col,
4. Keep an image matrix-shaped heatmap of entries into the output image matrix,
5. When all columns have been processed perform element-wise division of the image matrix and the image matrix heatmap.

3.2.13 Backpropagation of the convolutional layer using Im2col and Col2im

Since Im2col and its reverse operation Col2im have been defined, the backpropagation of the convolutional layer can be rewritten in the following way.

To obtain the filter kernel gradient:

1. X_{col} = Im2col of the last output of the convolutional layer,
2. $\delta W_{col} = \frac{\delta L}{\delta output} \times X_{col}$,
3. $\frac{\delta L}{\delta filters}$ = reshape δW_{col} back to kernel filter shape.

To obtain the input gradient:

- F_{col} = Im2col of the filters to produce the filter matrix,
- $\delta F_{col} = F_{col} \times \frac{\delta L}{\delta output}$,
- $\frac{\delta L}{\delta input}$ = Col2im of δF_{col}.

3.3 Prototyping

To design the best architecture for this application various kinds of ANNs were prototyped using TensorFlow, the industry-standard machine learning library. These prototypes were trained using the dataset that is described in the following implementation section. DNN, RNN and CNN prototypes were evaluated.

To make a fair comparison, similar numbers of total network trainable parameters were ensured for each architecture when comparing categories, this number was about 2×10^6. Additionally, the training was performed using a reduced dataset so that inferences can be made in a more reasonable amount of time and all of the models were trained for 10 epochs using the Adam optimizer to reduce convergence time. Furthermore, the ReLU activation function was used in all instances.

The DNN prototype consisted of an input layer with 256 inputs. This number was chosen because, sampling 128 IQ samples from an SDR, results in 256 numbers when it is considered as a sequence of real numbers rather than a sequence of complex numbers. The hidden layer has 4096 neurons and the output layer has 13 neurons as there are 13 classes. All layers except for the input layer have biases.

The RNN prototype consisted of one long short-term memories layer with 512 units, 2048 neurons in the hidden layer and 13 neurons in the output layer. The RNN was, however, eliminated from consideration because of its time complexity and known difficulty in training, the author was unable to successfully train an RNN prototype for this problem within the available prototyping time.

For the CNN models, the input is seen as a two-dimensional image with only one channel. The first CNN prototype (CNN1) consisted of one convolutional layer with 256 filters with kernels of 2×3, 128 neurons in the hidden layer and 13 neurons in the output layer.

The second CNN prototype (CNN2) consisted of two convolutional layers with 256 and then 80 filters with respective kernels of 1×3 and 2×3. There were 128 neurons in the hidden layer and 13 neurons in the output layer.

The third CNN prototype (CNN3) consisted of two convolutional layers with 256 and then 80 filters with respective kernels of 1×3 and 2×3. There was also one max pooling layer following every convolutional layer with kernels of 1×2 and 2×2 respectively. There were 128 neurons in the hidden layer and 13 neurons in the output layer.

	DNN	**CNN1**	**CNN2**	**CNN3**
Accuracy	0.37	0.42	0.83	0.47

Table 4.
Comparison of deep learning architectures

Considering table 4, it is clear that the so-named CNN2 architecture works the best on this problem, as is also suggested by [17]. Therefore the implementation section of this document will be focused on the CNN2 architecture for the final implementation.

It is interesting to note that max-pooling, as traditionally used in CNNs, degraded the performance. The author speculates the reason for this is because this is not a spatial scenario. To elaborate on that: in image processing, CNNs are used and the use of max-pooling makes sense because it condenses pixel activations over a spatial area. In the context of this problem, this does not hold, because a temporal sequence is being consolidated as a one channel visual input.

3.4 Implementation

3.4.1 Generating a dataset

The classifier algorithm is only one part of the requirements in designing and implementing a machine learning classifier system. Data is also required. Deep learning architectures work optimally when they are trained with large amounts of data. This presents a trade-off between training time and the ultimate quality of the model; if more and more samples are used to train the model the training time will grow linearly. Therefore it is also necessary to estimate an adequate middle-ground for dataset size.

The dataset for this system was generated using the GNU Radio API, as well as two SDRs. The RTL-SDR for receiving and the HACK-RF for transmitting. A Python script was written which interleaves the HACK-RF and the RTL-SDR, first transmitting and then receiving. Following that the received IQ samples are written to a comma-separated value (CSV) file

together with their correct class (modulation scheme).

GNU Radio is an open-source signal processing library that allows for the high-level operation of SDRs. This software package offers blocks that can be utilized in different configurations to accomplish the generation and transmission of various kinds of signals.

SDRs function through upconversion and downconversion of signals with assistance from a variable local oscillator. This means that functionally all signals ,going to and coming from an SDR, are baseband signals. This leads to the parameters that must be configured for the operation of an SDR: local oscillator frequency, bandwidth and gain. Given these parameters, a typical SDR such as the RTL-SDR will tune to the local oscillator frequency, and downconvert a bandwidth as specified. This results in a baseband signal that can be processed. The SDR transmission and reception procedures are illustrated in figures 10 and 11 respectively.

AM	**FM**	**PSK**	**QAM**	**FSK**
AMDSB	WBFM	2PSK	8QAM	2FSK
AMUSB-SC		4PSK	16QAM	GFSK
AMLSB-SC		8PSK	32QAM	GMSK

Table 5.
Modulation schemes considered

Table 5 indicates the groups and specific modulation schemes that are being considered. It is important to note that 4FSK and 8FSK as per the requirements and specifications have been replaced by GFSK and GMSK. The reason this was done is that 4FSK and 8FSK have largely fallen into obscurity in favour of more useful alternatives such as the mentioned GFSK and GMSK. The classifier was designed to be a 13 class classifier, meaning that every modulation scheme is represented individually. Ultimately a total of 680 000 samples were transmitted and recorded in multiple sessions. The actual training set was obtained by downsampling this dataset to 221 000 samples.

The validation set is obtained by the training-validation ratio during training time. This is usually selected to be 0.8, meaning from the dataset 80% of the samples will be used for training, and the remaining 20% will be used only to evaluate performance. The test set is a dataset completely different from the training set and validation sets. The test set must be sufficiently large such that an acceptable statistical figure can be accumulated for characterising the performance of the system. The test set was also downsampled from the original dataset and consists of 160 000 samples that do not occur in the training or validation sets.

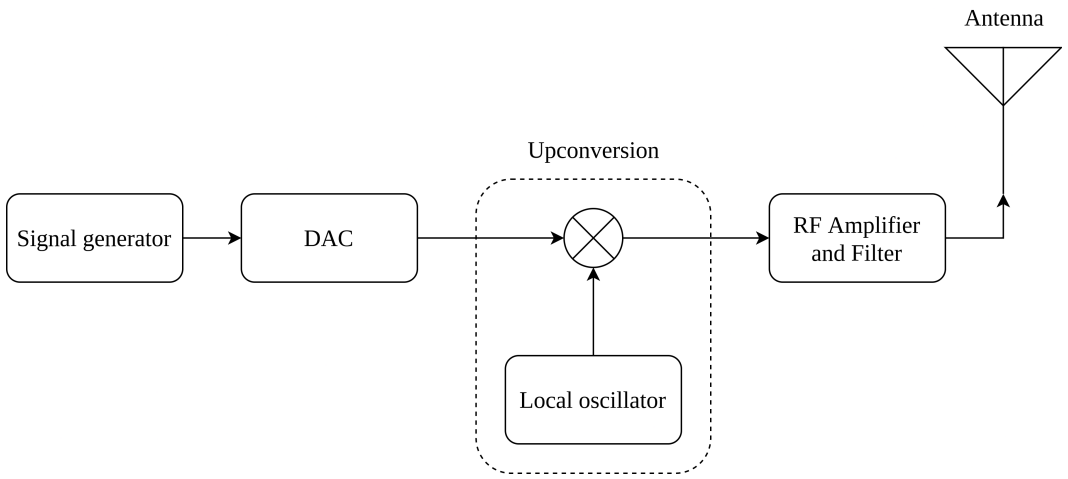


Figure 10.
Illustration of SDR signal transmission

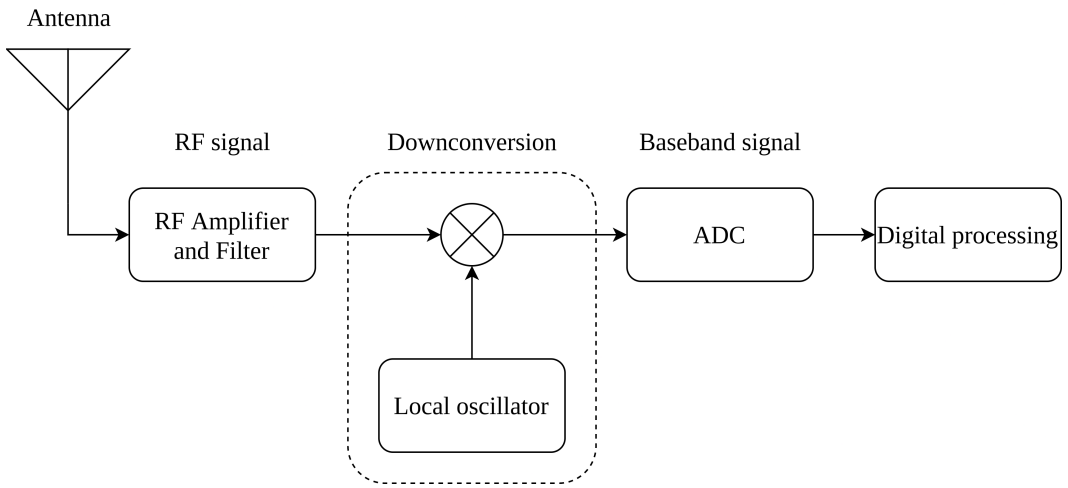


Figure 11.
Illustration of SDR signal reception

In GNU Radio there are blocks for FM, QAM, PSK, GFSK and GMSK. These existing blocks were used to generate their respective modulation schemes. For the remaining AMs and 2FSK no blocks exist, hence they were developed mathematically from scratch. In all analogue modulation schemes, audio files of the standard audio sampling rate of 44.1 kHz were transmitted; these included but were not limited to classical music, rock music, a man speaking and piano music. The digital modulation schemes were fed with the output of a pseudo-random number generator (PRNG) set to a suitable rate as required.

AMDSB was implemented by returning to the mathematical definition of AM. The steps as per the equation for AM was implemented to produce the amplitude modulated signal. An important point of this process is that the audio files were required to be interpolated to match the carrier frequency, which causes frequency copies of the signal. These frequency copies

were then filtered out with a GNU Radio *lowpass filter* block such that only one remains. AMDSB was generated using the abovementioned method of constructing the mathematical description of AM from GNU Radio operator blocks such as *add* and *multiply*. AMUSB-SC and AMLSB-SC were achieved by generating AMDSB and filtering the signal such that only either the lower or upper sideband remains.

2FSK was implemented also by returning to the mathematical definition of the modulation scheme. This translates to interleaving two oscillators depending on if a 1 or a 0 is required. In GNU Radio this was accomplished by connecting a PRNG outputting bits to a *voltage controlled oscillator* block. This was then upconverted to the carrier frequency where it travels across the transmission line. A key parameter of interest for 2FSK is the deviation from the carrier frequency of the two symbol frequencies.

Figure 12 illustrates the program flow of the script that governs the collection of the dataset. It is important to note that the 1000ms delay is included after starting the transmission to ensure the HACK-RF has enough time to start transmitting and settle into a steady state. This is an important step to ensure that good quality samples are recorded.

The signal parameters are selected from their respective ranges using a uniformly distributed PRNG in the Python script. The carrier frequency ranges are as per the project proposal in part 2 of this document, noting that the upper limit has been lowered by 50 MHz because the reliability of the upper limit varies from device to device when it comes to low-cost devices such as the RTL-SDR. The particular RTL-SDR that was used is unable to receive signals higher in frequency than 1700 MHz. The gain parameter on the RTL-SDR receiver is always set to 0 dB when generating a dataset. The symbol rate is fixed.

The gain parameter referred to in table 6 is the HACK-RF intermediate frequency (IF) gain. The HACK-RF also has another transmit gain parameter called the RF gain, this is however either on or off and to transmit over an RF cable it is set to off. Samples per symbol (SPS) only applies to digital modulations and describe the number of samples the same symbol occupies, this was kept static. Excess bandwidth (BW) is related to the root-raised-cosine (RRC) filter used in digital communications. Frequency deviation describes the difference between the carrier frequency and the symbol frequencies in 2FSK.

	Range
Carrier frequency	90 MHz – 1.7 GHz
Gain	0 dB – 47 dB
Audio	5 music files
Excess bandwidth	0.1 – 1.0
Frequency deviation	5 MHz – 30 MHz

Table 6.
Modulated signal generation parameters

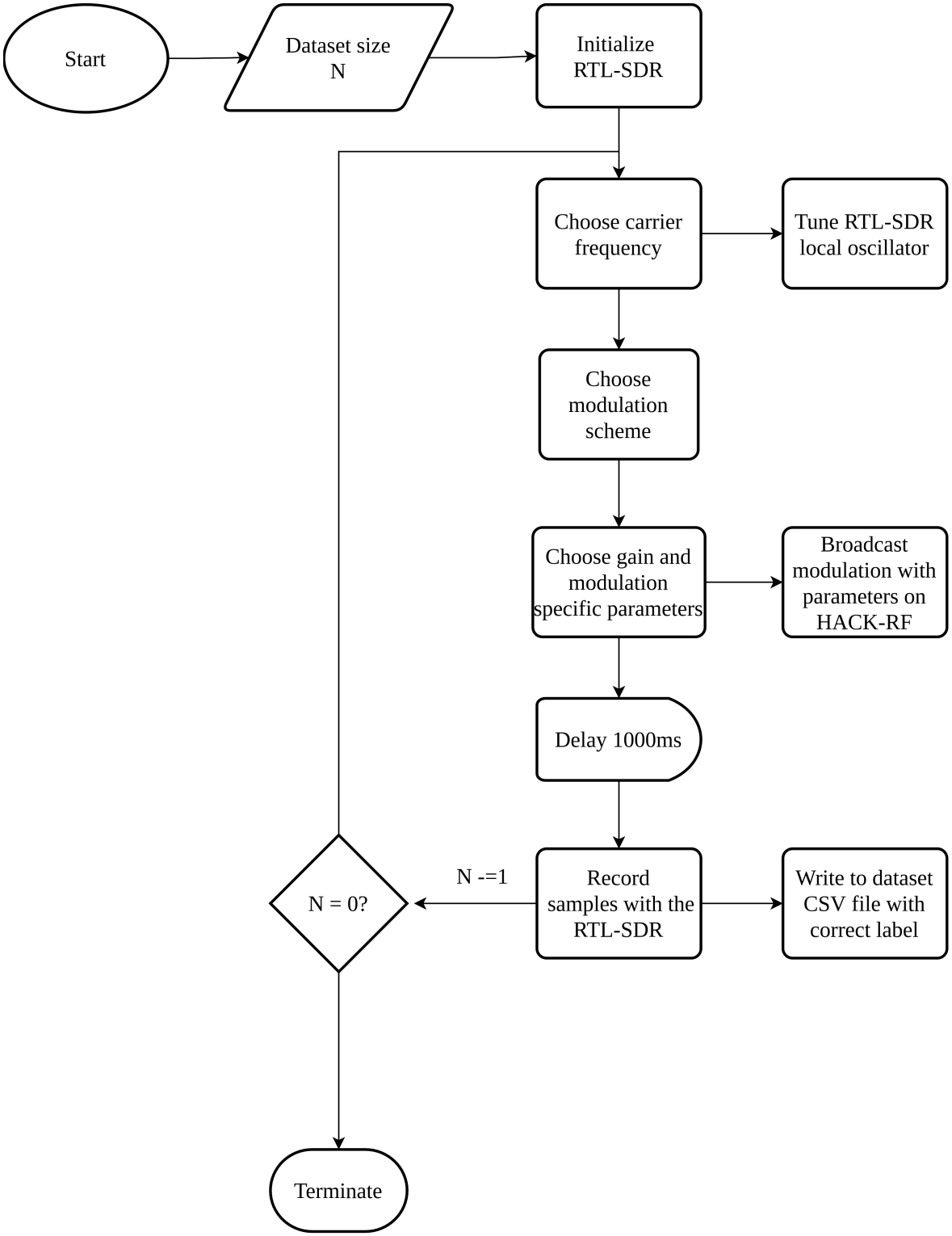


Figure 12.
Dataset collection script flowchart

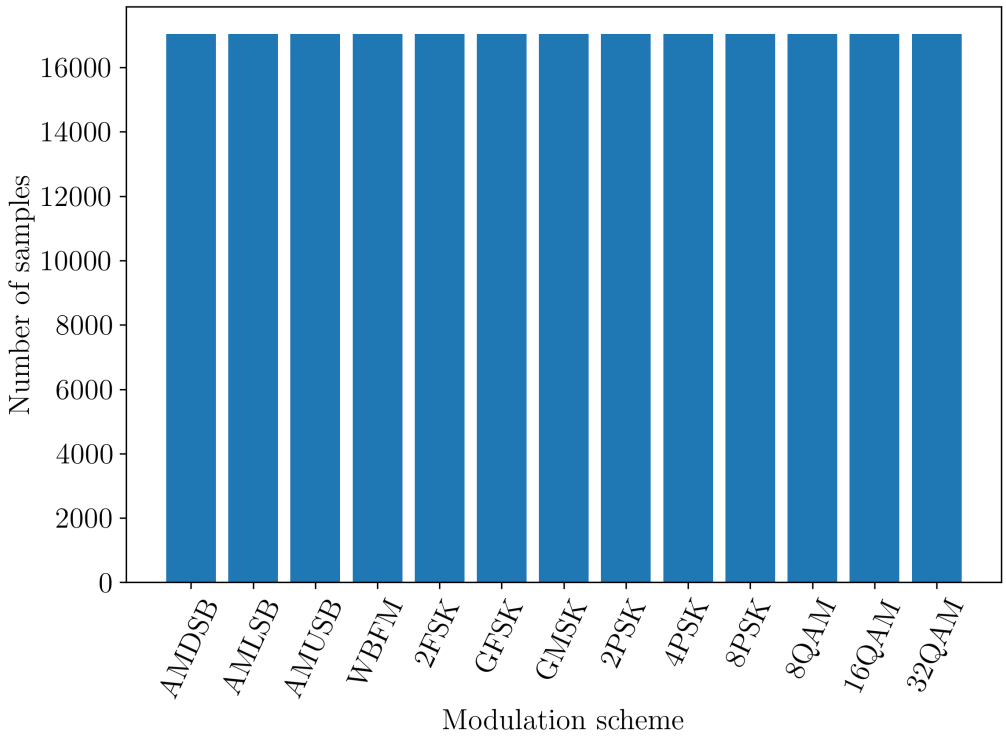


Figure 13.
Dataset modulation distribution

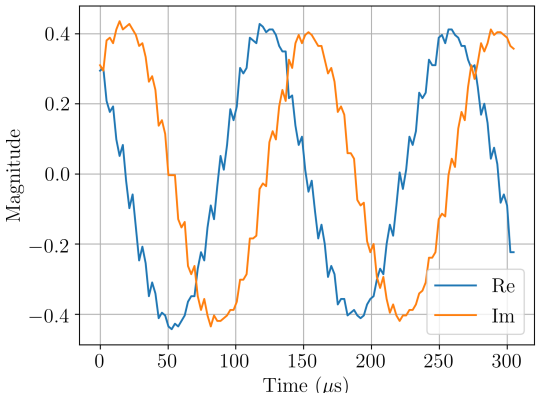


Figure 14.
Recorded AMDSB

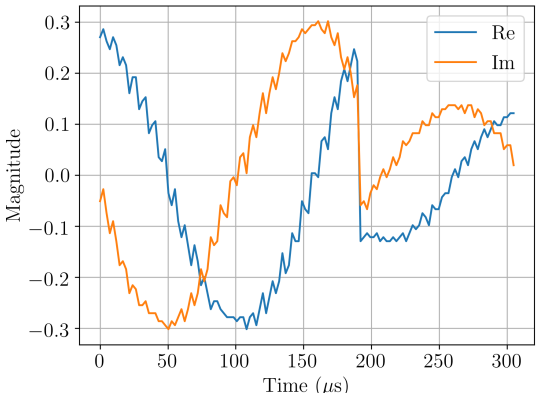


Figure 15.
Recorded AMLSB

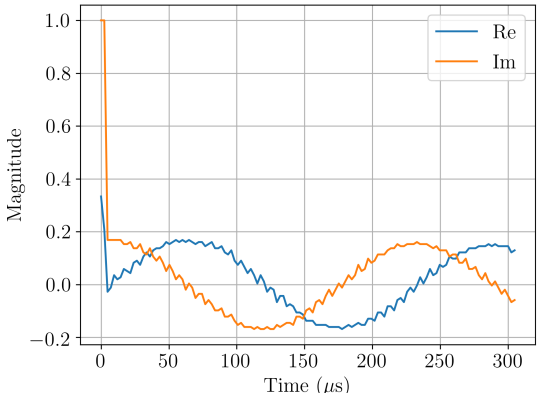


Figure 16.
Recorded AMUSB

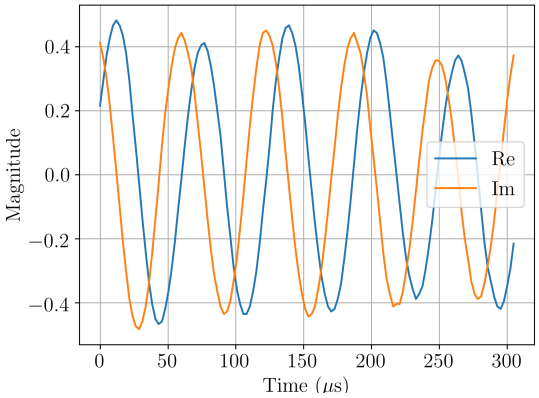


Figure 17.
Recorded WBFM

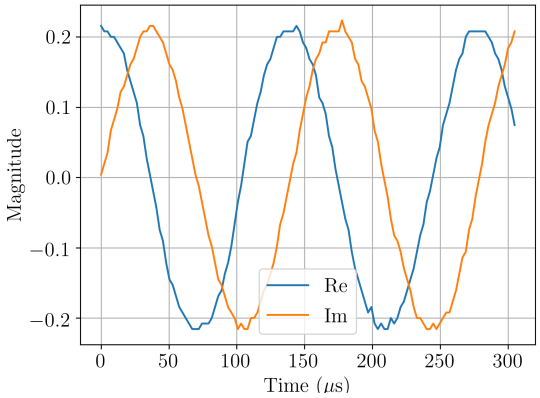


Figure 18.
Recorded 2FSK

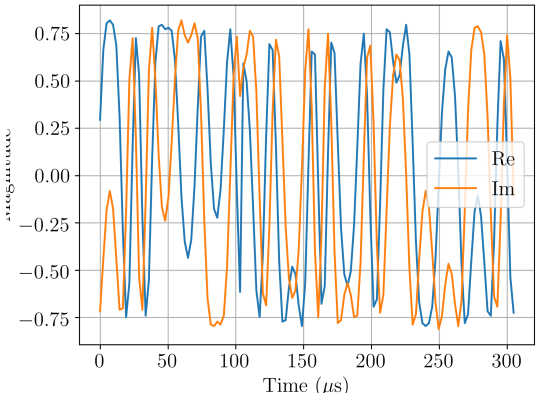


Figure 19.
Recorded GFSK

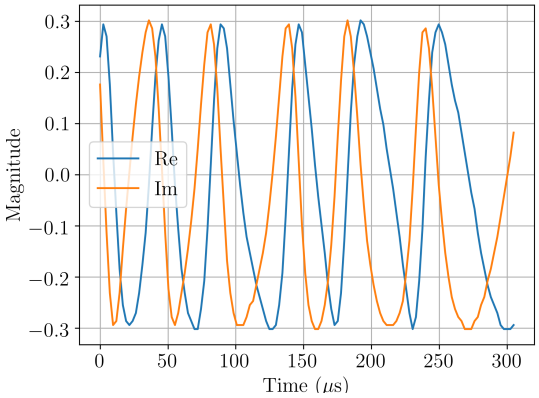


Figure 20.
Recorded GMSK

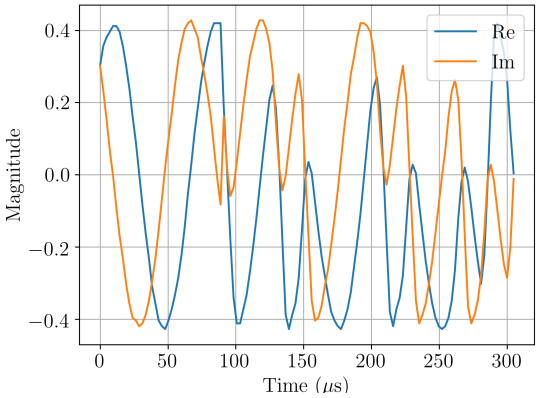


Figure 21.
Recorded 2PSK

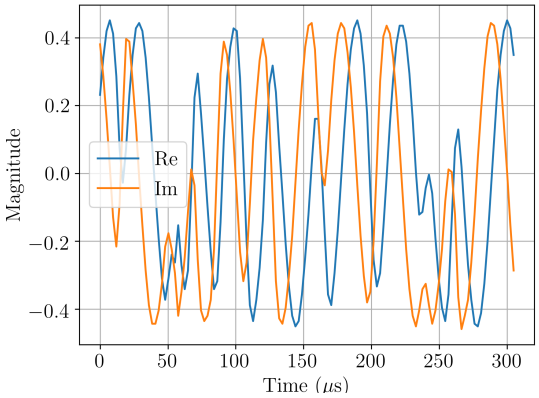


Figure 22.
Recorded 4PSK

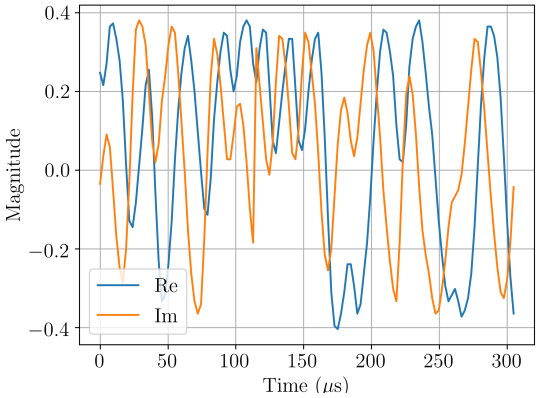


Figure 23.
Recorded 8PSK

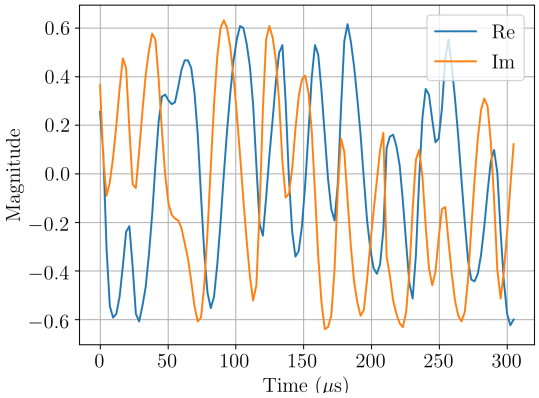


Figure 24.
Recorded 8QAM

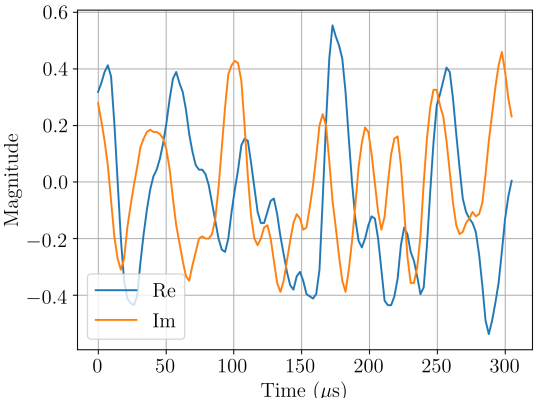


Figure 25.
Recorded 16QAM

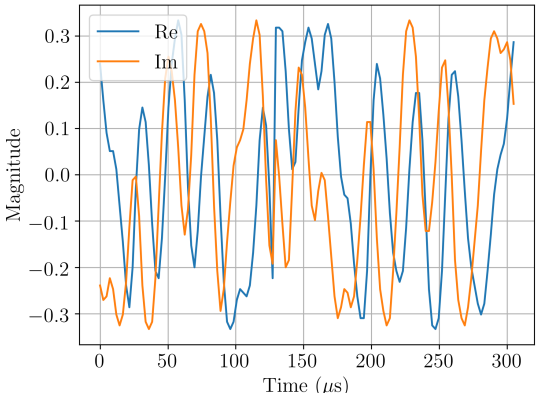


Figure 26.
Recorded 32QAM

3.4.2 CNN implementation

The CNN's final design and structure was guided by the literature and prototyping detailed in this document. Table 7 summarizes the structure of the final CNN. Figure 27 visualizes the structure. The architecture is similar to that which was proposed in [17], as this was also corroborated by the results of prototyping different structures. As stated previously, when reading samples from an SDR it presents in the form of complex IQ samples. In this model, 128 IQ samples are read from an SDR; subsequently, the real and imaginary parts are interpreted as two columns of a 128-long matrix.

	Input shape	Activation	Filter shape	No. of filters	No. of neurons
Convolutional layer 1	2×128	ReLU	1×3	256	–
Convolutional layer 2	$2 \times 126 \times 256$	ReLU	$256 \times 2 \times 3$	80	–
Dense layer 1	1×9920	ReLU	–	–	9920
Dense layer 2	1×9920	ReLU	–	–	128
Output layer	1×128	Softmax	–	–	13

Table 7.
Structure of designed CNN

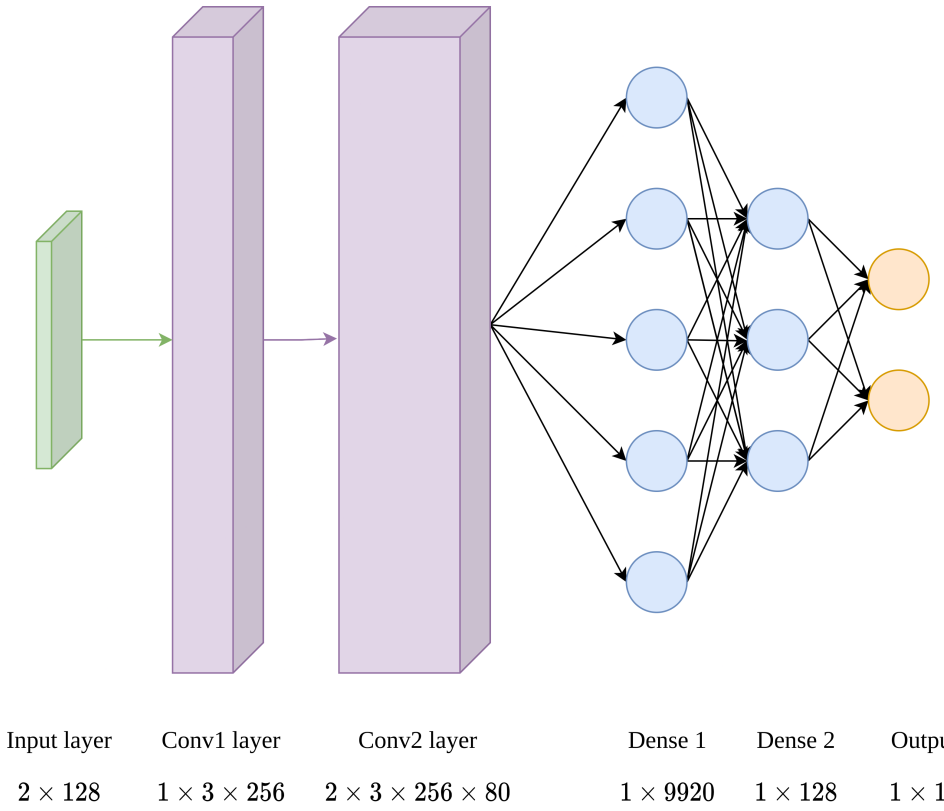


Figure 27.
Illustration of CNN architecture

To implement this structure in Python code a class was created with member functions that perform all of the many important functions that are required to make up this structure and train it. Figure 34 shows the Unified Modelling Language (UML) class diagram of the script that was produced.

In practical terms, there is a problem with the softmax function as it is mathematically defined. When it is implemented as is, it is subject to underflow and overflow errors. This is highly undesirable. To rectify this problem the softmax function can be rendered numerically stable. This is done by first subtracting the maximum of the input vector from every element of the input vector. Calculating the softmax function in this manner will never result in overflow, underflow or division by zero whilst preserving the output.

To ensure general numerical stability in a computer program where one might run into divisions by zero, a very small value can always be added to the denominator of any fraction. In this case, it was selected that $\varepsilon = 1 \times 10^{-8}$, such that any instance of division by zero is recoverable and does not terminate the program or its progress.

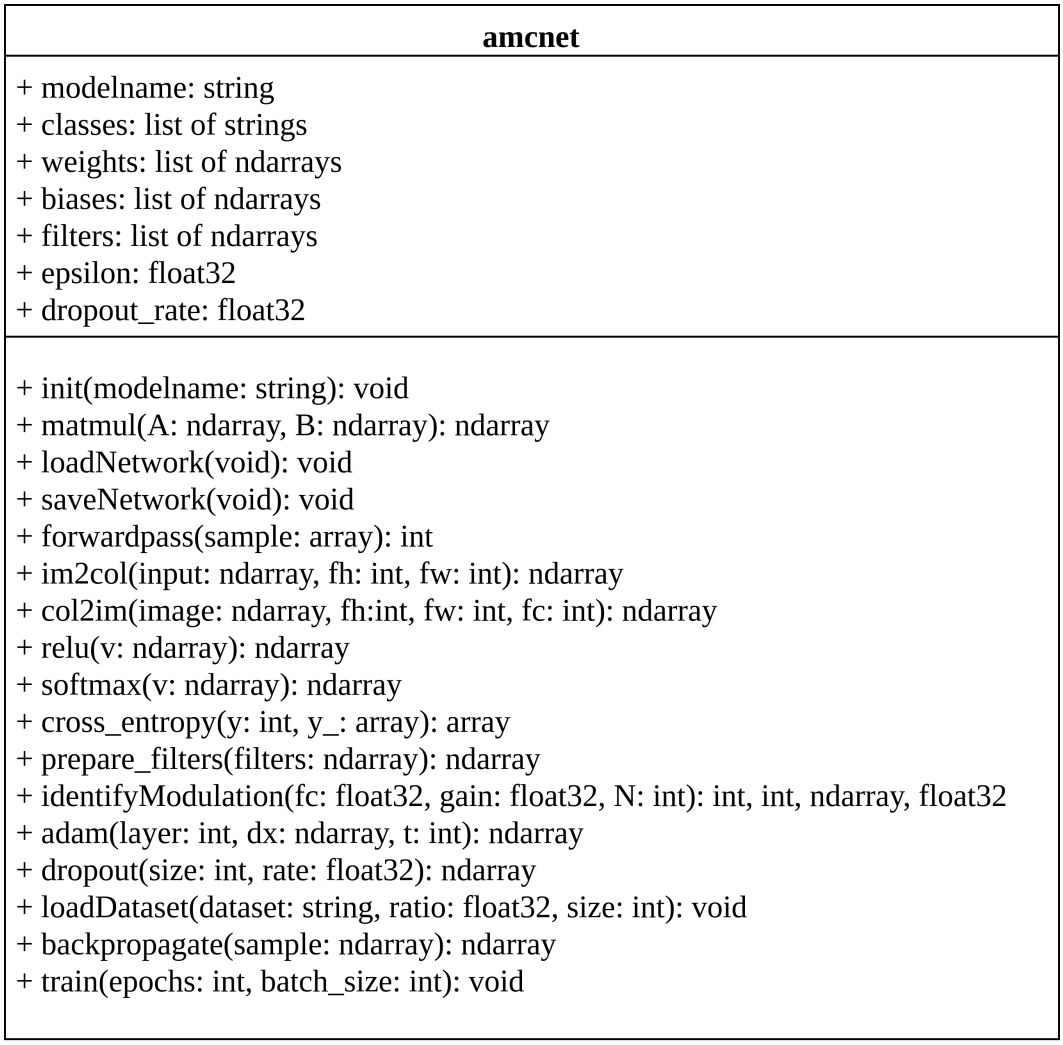


Figure 28.
UML class diagram of amcnet

A brief description is given of the utility of each of the functions in figure 34:

- **init()** - Instantiates the amcnet class, initializes the variables and loads the saved model with the same name as the input string.
- **matmul()** - Multiplies two input matrices and returns the resultant matrix. This function exists such that the system can still work even when no GPU is available.
- **loadNetwork()** - Load the weights, biases and filters stored in a *.npy* file.
- **saveNetwork()** - Save the weights, biases and filters to a *.npy* file.

- **forwardpass()** - Performs one forward pass of the whole structure and returns the resultant predicted class.
- **im2col()** - Executes the im2col transformation on the input.
- **col2im()** - Executes the col2im transformation on the input.
- **relu()** - Activates all input values using ReLU.
- **softmax()** - Calculates the softmax probability distribution of the input vector.
- **cross_entropy()** - Calculates the categorical cross-entropy loss for the input predicted class and actual class.
- **prepare_filters()** - Prepares the input filters for use in convolution by matrix multiplication by restructuring their form.
- **identifyModulations()** - Tunes the RTL-SDR to a specified carrier frequency and gain, then collects N samples, runs them through the classifier and returns the aggregated output.
- **adam()** - Computes the Adam optimizer algorithm.
- **dropout()** - Computes a dropout mask given the size and dropout rate of a dense layer.
- **loadDataset()** - Loads the specified dataset into memory.
- **backpropagate()** - Computes one full backpropagation of the structure given the input sample.
- **train()** - Trains the model for a specified number of epochs.

Some key functions and datatypes from the well-known Python library Numpy were used. The following list details the functions that were used and a brief description of their utility:

- **dot()** - Performs vector and matrix multiplication.
- **reshape()** - Transforms an N dimensional array into a different shape with the equivalent number of entries.
- **exp()** - Calculates the exponential function e^x.
- **sum()** - Calculates the sum over all the indices of the input.
- **array()** - Transforms other array datatypes to the Numpy array type.
- **expand_dims()** - Adds an empty dimension to an N dimensional array.
- **argmax()** - Returns the index of the maximum value found in the input.
- **zeros()** - Allocates memory for an N dimensional array of specified dimensions initialized with zeros.

- **ones()** - Allocates memory for an *N* dimensional array of specified dimensions initialized with ones.
- **random.randn()** - Draws a sample from a normal distribution with specified mean and variance.
- **float32** - Datatype for single precision floating point format.

The libraries pycuda and scikit-cuda were used to interface with GPU hardware using CUDA. This was implemented to outsource matrix multiplication to GPU hardware to speed up the execution of the program. The following details a list of functions that were used to enable this approach:

- **skcuda.linalg.dot()** Performs matrix multiplication of two matrices stored in GPU memory.
- **skcuda.gpuarray.to_gpu()** Writes array to GPU memory from main memory.

From the numexpr library the **numexpr.evaluate()** function was used in places to facilitate multiprocessing where possible. This library is called numpy express, because it sports parallel implementations of many numpy functions, including the evaluation of expressions.

3.5 Optimization of parameters

3.5.1 Parameter initialization

Since the neural network consists of many weights and biases, and in the case of CNNs also filter kernels, these parameters must assume starting values that will minimally cripple the learning process. Instead of attempting to run simulations to determine the best scaling value for parameter initialization, a statistically derived method was used.

He initialization was first proposed in [25], where the authors details an approach to initialize the weights and biases in layers that make use of the ReLU activation function. The approach is to sample initialization values from a Gaussian distribution

$$N\left(0, \sqrt{\frac{2}{N}}\right) \tag{32}$$

Where N is the number of neurons in a layer. In a convolutional layer, the total number of neurons is the number of weights in each filter kernel multiplied by the number of filters. One heuristic for judging the quality of initialization is if all possible outputs are more or less equiprobable directly following initialization.

3.5.2 Learning rate

There is a delicate balance to consider when selecting a learning rate. If the learning rate is too small, the model will take too long to converge. A small learning rate also risks getting stuck in a saddle point; a saddle point is a minimax point on the surface of the error function where the gradient in all directions is zero, effectively halting the convergence of the algorithm. When the learning rate is too large the algorithm will almost certainly jump over local minima, also resulting in divergence.

The test conducted to determine the best learning rate for the CNN model involved running a fixed number of epochs of the model using the reduced dataset. During this test, no regularization was implemented. The results are plotted in figure 29.

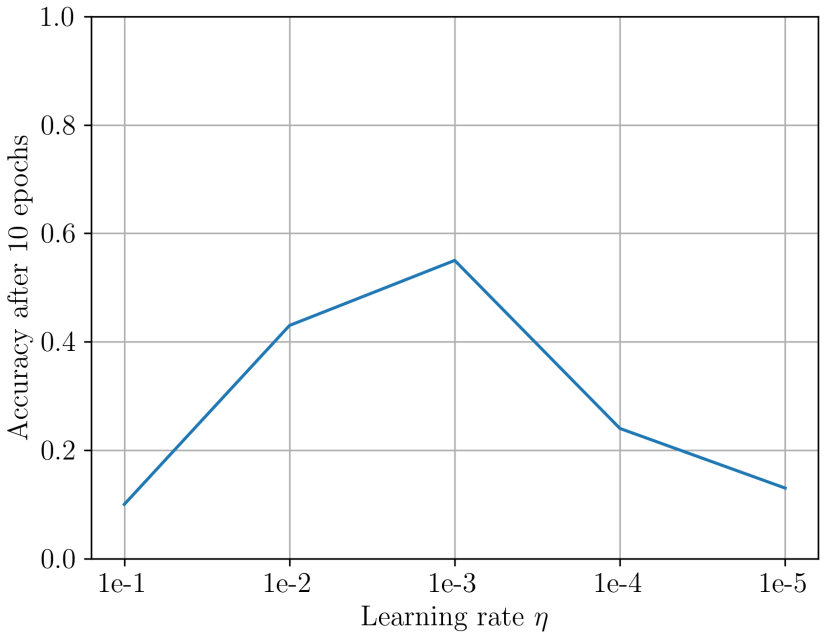


Figure 29.
Learning rate test

From figure 29, it was concluded that the best learning rate is $\eta = 1 \times 10^{-3}$.

3.5.3 Batch size

Batch size is the number of samples that are backpropagated before the aggregated deltas are applied to the parameters. Generally, the larger the batch size is the better the generalization performance becomes. This is only true up to a recommended limit of 256, often it is strictly recommended to not exceed 32. A batch size of 1 is not being considered, because for the

batch size to begin to provide increased generalization and faster training it needs to be larger than one.

The test to determine the best batch size was conducted by running the model for a fixed number of epochs on the reduced training set at different batch sizes. During this test, no regularization was implemented. The results are plotted in figure 30.

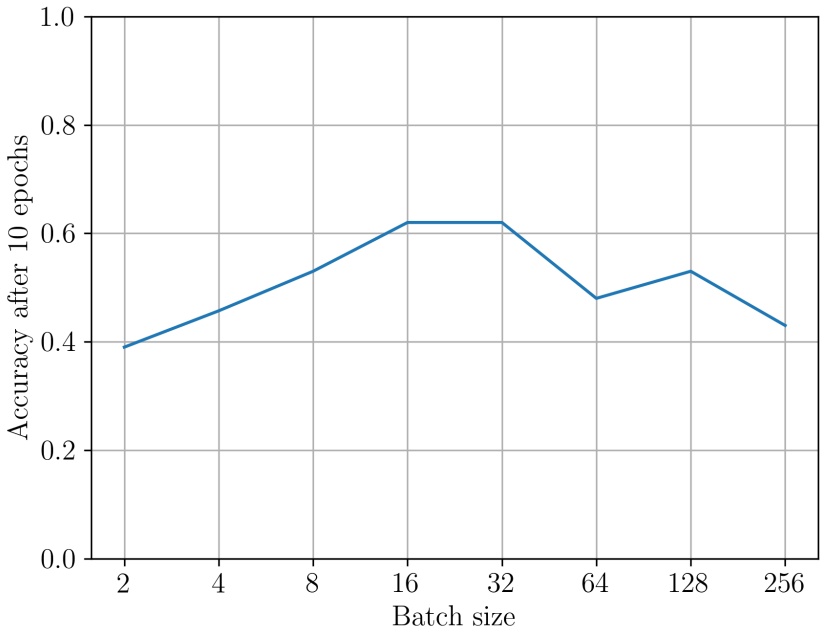


Figure 30.
Batch size test

From figure 30, it was concluded that the best batch size is 32. Even though batch sizes of 16 and 32 perform the same, the latter will incur fewer runs of the optimizer algorithm thus reducing training time.

3.5.4 Dropout

The dropout parameter p represents the probability that a neuron will be deactivated in any particular update step. Any dropout parameter is known to improve the validation accuracy of models. However, there is a trade-off between the dropout parameter and training time. If the dropout parameter is too high, the algorithm will take much longer to converge, if at all.

The test to determine the best dropout rate p was conducted by running the model for a fixed number of epochs on the reduced training set at different dropout rates. The results are plotted in figure 31.

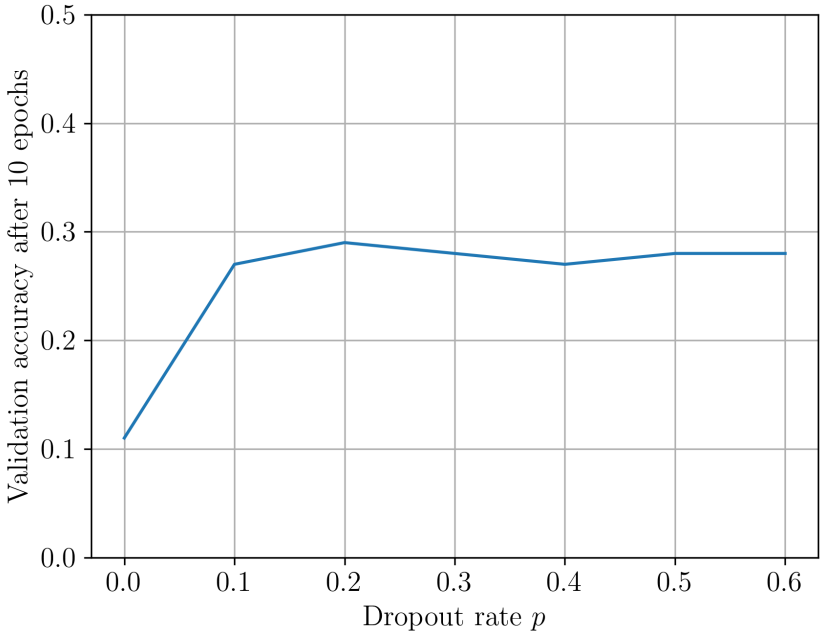


Figure 31.
Dropout parameter test

From figure 31, it was concluded that the best dropout rate $p = 0.1$. The reason this was chosen is that it is the most appropriate balance between increased generality and training time, the returns diminish strongly from $p = 0.1$ on.

3.5.5 Measurement consolidation

For every input sequence of 128 SDR samples, the model produces a probability distribution over all 13 possible classes. Using multiple measurements to produce an output provides the opportunity to increase the performance of the system. The classes that are detected correctly more than 50% of the time will sport an increased probability of correct detection.

This was done by taking N successive sequences of 128 samples from the RTL-SDR, then running them all through the model to produce class probability distributions. Then the mean of all N probability distributions can be computed. The maximum value of the mean distribution gives the predicted class.

It was anticipated that there is a point of diminishing returns for increasing the number of measurements that are consolidated N. A test was conducted to find this value which benefits the operation of the system optimally. This was done by operating the system in real-time while varying the number of measurements taken and recording the results.

In this test it was concluded that the returns diminish after 4 samples have been computed. This is because the correct detections of classes starting with a validation accuracy of at least 50% increase substantially, however, for the other classes the opposite is true after 4

samples. Consolidating more measurements than this will not increase the probability of correct detection. It is not practical to keep increasing the number of measurements taken, hence it should be chosen as the point of diminishing returns of accuracy.

3.5.6 Signal to noise ratio estimation

Estimating signal to noise ratio (SNR) can be a challenging task in a practical setting. It is necessary to estimate the SNR of the signal that is going into the classifier because the system is to work only above an SNR of 20 dB.

Given I and Q from complex samples, then the root mean square (RMS) voltage value of the signal can be computed from

$$V_{RMS} = \sqrt{\frac{I^2 + Q^2}{2}} \tag{33}$$

This process can be repeated to establish a measurement for both the signal and the noise floor. Subsequently the SNR in dB then is given by

$$\text{SNR} = 20 \times \log_{10}\left(\frac{V_{RMS}^{signal}}{V_{RMS}^{noise}}\right) \tag{34}$$

To obtain the estimated SNR for a sample sequence of 128 IQ samples, this process is repeated for every sample. The results are averaged to produce an estimated SNR for the sample set.

3.5.7 Training

The model was trained for 15 epochs on the training set using the parameters as per the parameter optimization section. Figure 32 presents the training curves over epochs.

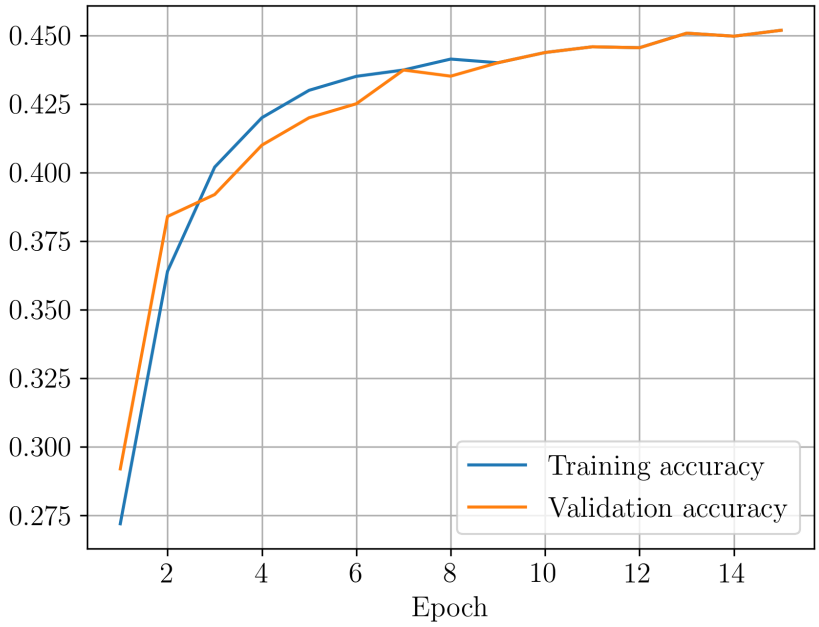


Figure 32.
Training of AMC CNN model

3.5.8 Model performance evaluation

The performance of the trained model was evaluated using the test dataset. A useful way to visualize the performance of a classifier algorithm is to use a confusion matrix. A confusion matrix showcases the relationship between the predicted classes and the actual classes. In effect, it shows how often one class is confused for another.

Figure 33 shows the confusion matrix for the test set. This represents the general performance of the model. This is true for three reasons: firstly, the data in the test set has never been seen before by the model. Secondly, it is sufficiently large. Lastly, it is comprised of a uniformly random sampling of signal modulation types and parameters.

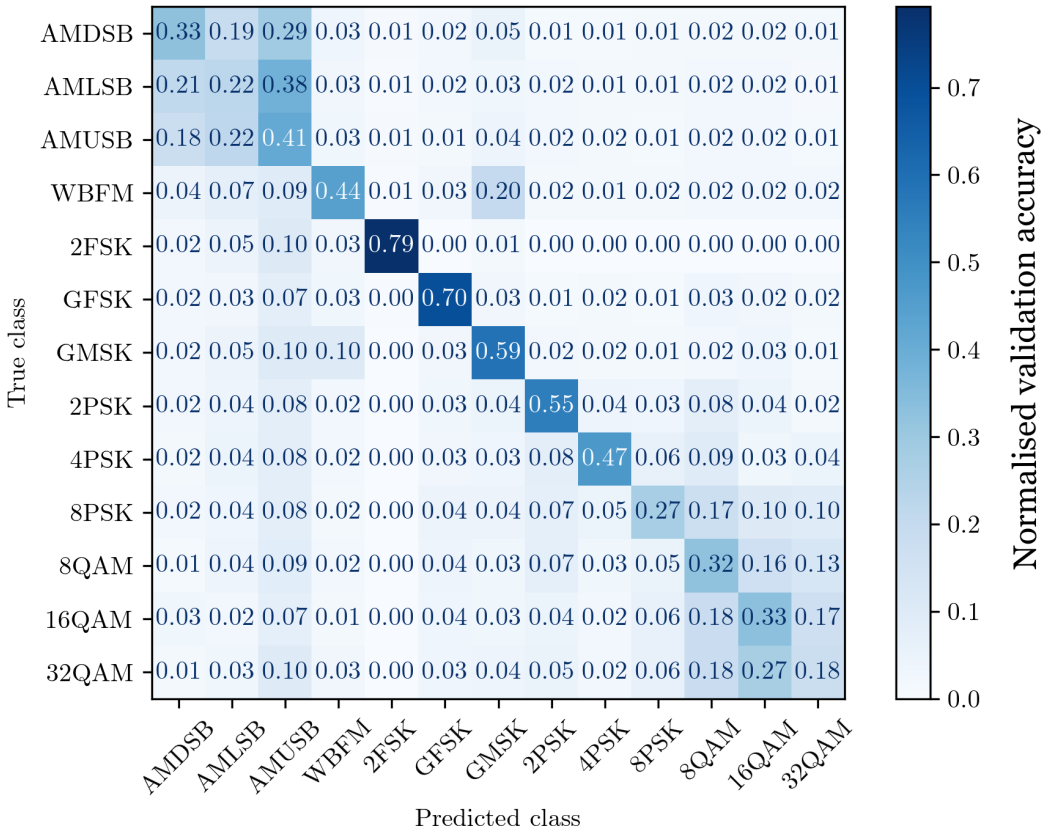


Figure 33.
Normalized confusion matrix of test set

3.5.9 AMC system implementation

Hardware
The AMC system was implemented on the Nvidia Jetson Nano Dev Kit v3. This platform was selected due to its unique onboard Nvidia Maxwell GPU with 128 processing cores. The reason this quality matters for this project is because the CNN was implemented such that all major operations have been reformed to be matrix multiplications. GPU hardware is designed to be exceedingly fast with parallel processing matrix operations.

The Jetson Nano has four universal serial bus (USB) 3.0 ports. This interface was used to connect to the RTL-SDR, which uses USB 2.0. The Python wrappers for the RTL-SDR driver are used to communicate with the device.

The Jetson Nano does not have a chip for wireless communications built-in. It does however have an ethernet port. An interface with the Jetson Nano can be established by connecting this ethernet port to another computer or to a network that supports dynamic host configuration protocol (DHCP). In the former case it would be necessary to assign internet protocol (IP) addresses manually.

Software
The following list details the software packages that were used in the implementation of the AMC system:

- Python 3.6.9
- Numpy 1.13.3
- Numexpr 2.7.3
- PyCUDA 2021.1
- Scikit-CUDA 0.5.3
- PyRTLSDR 0.2.92
- Termcolor 1.1.0
- GNU Radio 3.8.1.0

User interface
The user interface for the AMC system was made command line-based. Accessing the Jetson Nano is accomplished through Secure Shell (SSH). This is done with the following command:

```
$ ssh amc@XXX.XXX.XXX.XXX
```

Where *amc* is the user account on the Jetson Nano and the redacted text represents the IP address of the Jetson Nano on the network it is connected to. The password was set to be 123, as this setup is for educational and demonstration purposes only. No security is required.

Once an SSH terminal to the Jetson Nano has been established, to run the AMC system, the following command should be used:

```
$ python3 amc.py modelname f_c gain bw N
```

This command executes the AMC system script with a number of command-line parameters that influence the operation of the system. The *modelname* parameter represents the file of weights, biases and filters that is loaded. The following parameters represent the centre frequency, gain, bandwidth and number of measurements that the RTL-SDR will be configured to in the program run. The system should be operated at the full bandwidth of 2.4 MHz.

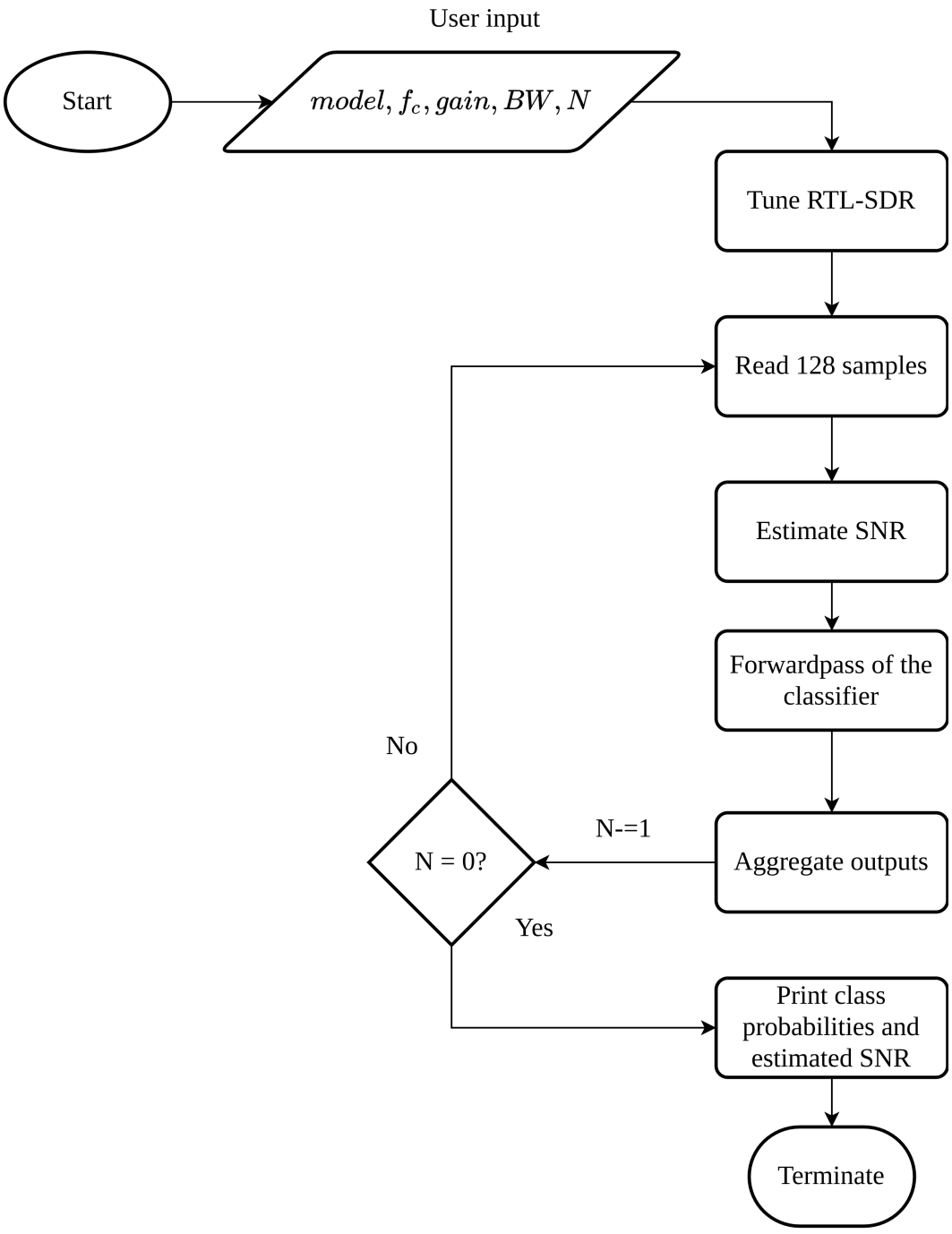


Figure 34.
Flowchart of AMC system

4. Results

This section pertains to the measured results obtained from the AMC system.

4.1 Summary of results achieved

Intended outcome	Actual outcome	Location in report
Core mission requirements and specifications		
Identify modulation schemes: AMDSB, AMUSB, AMLSB, WBFM, 2PSK, 4PSK, 8PSK, 8QAM, 16QAM, 32QAM, GFSK, GMSK and 2FSK.	The system is able to identify all of the required modulation schemes.	Section 4.2.1
The system must operate above a minimum accuracy specification of on average 75% correct detections for the identification of the prescribed modulation schemes.	The system identifies modulation schemes correctly on average at a rate of 69% above 20 dB.	Section 4.2.1
The system will operate on the FM radio, TV, GSM and ISM frequency bands between 90 MHz - 1750 MHz with no signal overlapping in the band of interest.	The system is able to identify modulations across the frequency range.	Section 4.2.2
The execution time of the entire AMC pipeline must be reduced such that the potential action can be taken immediately.	The execution time of the system was reduced by employing coding techniques and parallel processing libraries. The time for run of the system is 0.186s on average.	Section 4.2.3
Field condition requirements and specifications		
The system will operate under normal indoor and outdoor conditions for average minimum and maximum temperatures in Pretoria specified as the range from 5°C to 30°C.	The system was tested in the temperature range of 15°C to 34°C. In this range the system was fully operational. Lower temperatures were not tested.	-

The system will operate in a dry environment, protected from moisture and in absence of contact with any kind of liquid.	The system is not waterproof and was not tested in the presence of any liquids or significant moisture. The system was exclusively tested and operated in dry environments.	-

Table 8.
Summary of results achieved

4.2 Qualification tests

4.2.1 Qualification test 1: Classification accuracy

Objectives of the test or experiment
The objective of this qualification test is to measure the classification accuracy of the system at an SNR of 20 dB.

Equipment used

- RTL-SDR
- HACK-RF One
- Nvidia Jetson Nano Dev Kit v3
- Coaxial cable
- Ethernet cable
- HP Envy 13 laptop (PC1)

Test setup and experimental parameters
PC1 is connected to the HACK-RF with a USB cable and the Jetson Nano is connected to the RTL-SDR with a USB cable. The Jetson Nano and PC1 are connected together with the ethernet cable such that PC1 can SSH into the Jetson Nano.

Steps followed in the test or experiment

1. The AMC system is set to run on loop at a fixed frequency of 90 MHz.
2. The student broadcasts an RF modulated signal using PC1 with random parameters, fixed frequency of 90 MHz and at an SNR of approximately 20 dB.

3. The sequence of predicted classes are written to a file.
4. Steps 2 to 3 are repeated.
5. The results are visualized using a confusion matrix.

Results
In total 100 measurements of each class were collected. The result of this experiment is represented by figure 35.

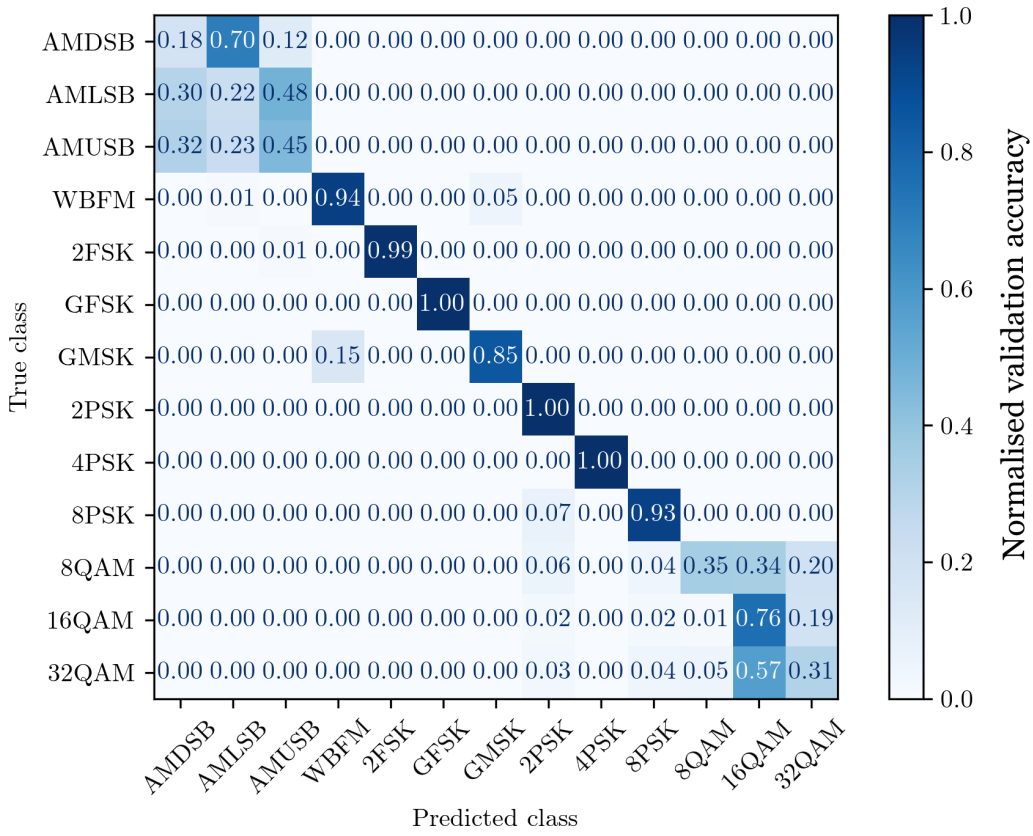


Figure 35.
Normalized confusion matrix of AMC system for 20 dB

The average accuracy can be calculated by taking the mean of the diagonal of the confusion matrix. The average accuracy of the system across all 13 classes at a SNR above 20 dB is 69%.

4.2.2 Qualification test 2: Frequency range operation

Objectives of the test or experiment
The objective of this qualification test is to measure the classification accuracy of the system over the full frequency range of 90 MHz to 1700 MHz.

Equipment used

- RTL-SDR
- HACK-RF One
- Nvidia Jetson Nano Dev Kit v3
- Coaxial cable
- Ethernet cable
- HP Envy 13 laptop (PC1)

Test setup and experimental parameters
PC1 is connected to the HACK-RF with a USB cable and the Jetson Nano is connected to the RTL-SDR with a USB cable. The Jetson Nano and PC1 are connected together with the ethernet cable such that PC1 can SSH into the Jetson Nano.

Steps followed in the test or experiment

1. The AMC system is set to run on loop at a certain frequency.
2. The student broadcasts an RF modulated signal using PC1 with random parameters at the aforementioned carrier frequency, at an SNR of approximately 20 dB.
3. The sequence of predicted classes are written to a file.
4. Steps 1 to 3 are repeated.
5. The results are visualized using a confusion matrix.

Results
Three points in frequency were tested: the low extreme of 90 MHz, the high extreme of 1700 MHz and the middle point of 895 MHz. At each point in frequency, 100 samples of each of the 13 classes were recorded.

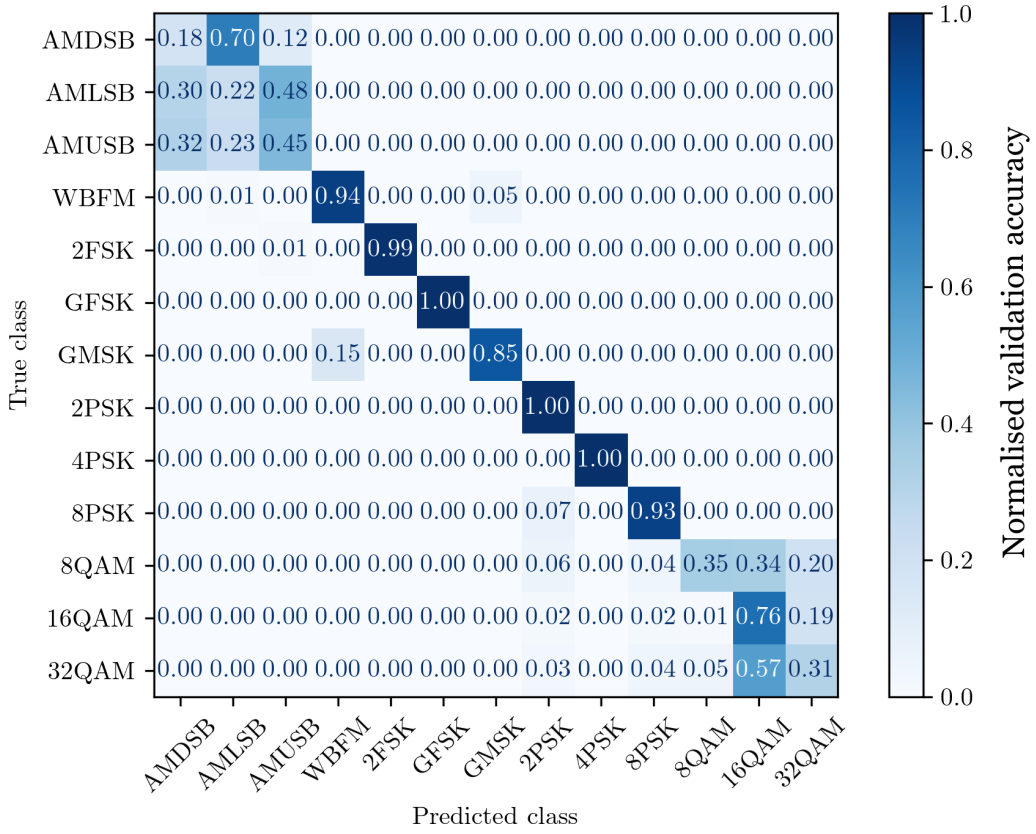


Figure 36.
Normalized confusion matrix of AMC system for 20 dB at 90 MHz

The average accuracy is calculated by taking the mean of the diagonal, resulting in 69%.

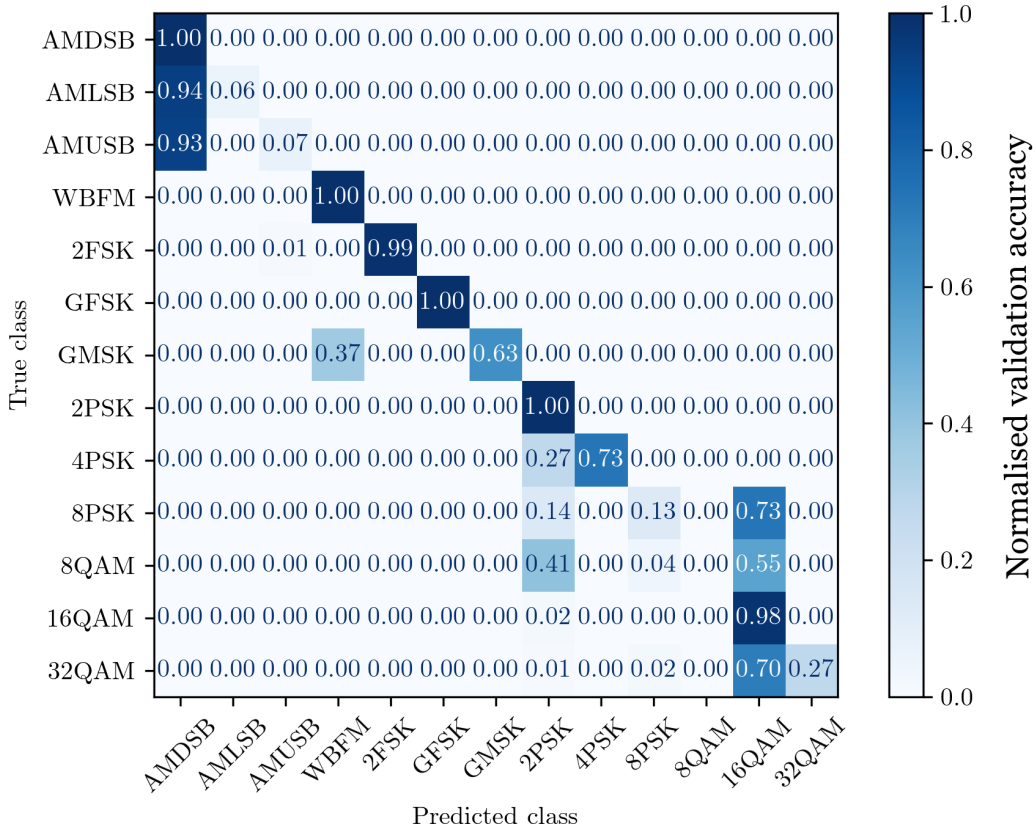


Figure 37.
Normalized confusion matrix of AMC system for 20 dB at 895 MHz

The average accuracy is calculated by taking the mean of the diagonal, resulting in 60%.

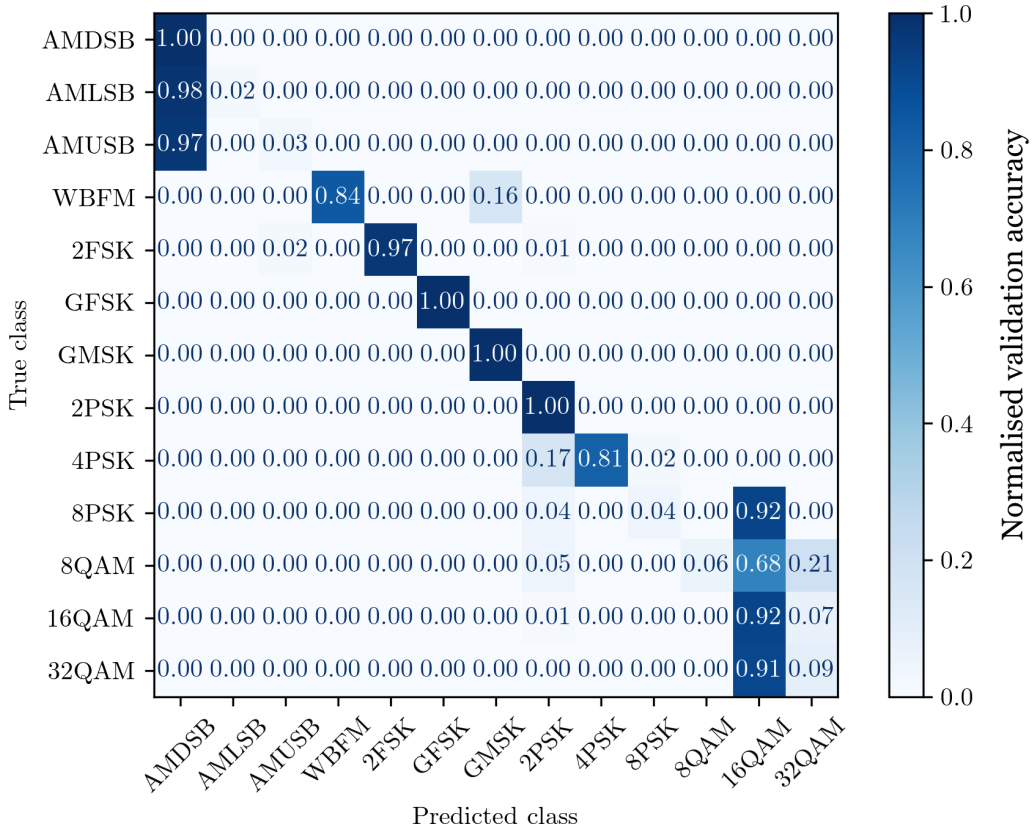


Figure 38.
Normalized confusion matrix of AMC system for 20 dB at 1700 MHz

The average accuracy is calculated by taking the mean of the diagonal, resulting in 59%.

4.2.3 Qualification test 3: Execution time

Objectives of the test or experiment
The objective of this test is to determine if the system produces results within the specified 5s of being executed.

Equipment used

- RTL-SDR
- HACK-RF One
- Nvidia Jetson Nano Dev Kit v3
- Coaxial cable
- Ethernet cable

- HP Envy 13 laptop (PC1)

Test setup and experimental parameters
PC1 is connected to the HACK-RF with a USB cable and the Jetson Nano is connected to the RTL-SDR with a USB cable. The Jetson Nano and PC1 are connected together with the ethernet cable such that PC1 can SSH into the Jetson Nano.

Steps followed in the test or experiment

1. The AMC system is set to run on loop at a fixed frequency of 90 MHz receiving with the RTL-SDR.
2. PC1 broadcasts random modulations with random parameters and a fixed frequency of 90 MHz using the HACK-RF.
3. A function times every loop of the AMC system and this value is written to a file.
4. Step 3 is repeated.
5. The arithmetic mean and variance of the measurements are calculated.

Measurements
The following result was obtained by repeating the described process 10000 times. Every time the execution time was measured, following that the mean and variance was calculated and tabulated in table 9.

Mean	Variance
0.186s	$25\mu s$

Table 9.
Statistical results

5. Discussion

5.1 Interpretation of results

The results that were achieved by the AMC system do not completely satisfy the requirements. The system achieved a 69% classification accuracy across all 13 classes at an SNR of 20 dB. This falls short of the 75% requirement by 6%.

In figure 32 it can be seen that the model trained for 15 epochs only. The reason for this is that the model consistently encountered a local minimum point on the surface of the loss function. This prevented further training and subsequently crippled the ultimate potential of the system. It is noteworthy that the validation accuracy followed the training accuracy very closely, this is indicative of the model generalizing very well. The student pursued many different strategies to subvert this local minimum such as varying the starting learning rate and experimenting with simulated annealing of the learning rate. In every epoch, the samples were taken from the training set in a pseudo-random order to introduce noise into the system to discourage convergence on local minima. The model continued to converge on a local minimum. Despite the model only training to a validation accuracy of 43%, in the implementation, it notably still facilitates a real signal detection accuracy of 69% at 20 dB.

The normalized confusion matrix for the AMC system is given in figure 35. Analysing this graph it is clear that the system performs very well on the following modulations: WBFM, 2FSK, GFSK, GMSK, 2PSK, 4PSK, 8PSK and 16QAM. There is a slight confusion between WBFM and GMSK. The reason for this is because spectrally GMSK and WBFM are very similar.

The system does not perform well on the modulations within the AM group. This is thought to be because AMLSB and AMUSB are directly obtained from AMDSB, making the members of the group difficult to distinguish from one another. Spectrally, AMUSB and AMLSB are reflections of each other, and both of them can be found in the spectrum of AMDSB. However, it is noteworthy that the AM group is fully distinguished from the rest of the modulation schemes. That is to say, even if the system is not able to identify precisely what type of AM is present, it will still identify with confidence that AM is present.

The system also does not perform well within the QAM group of modulation schemes. Significantly, 16QAM stands out as being identified correctly much more often than 8QAM and 32QAM. In the confusion matrix shown in figure 35, it is observed that 8QAM and 32QAM are often mistaken for 16QAM. The reason for this is thought to be because 16QAM has more symbol overlap with 8QAM and 32QAM than they do with each other. It is important to note that figure 35 also clearly indicates the QAM group is completely distinguished from the other groups.

The system is able to identify modulations schemes across the full specified frequency range at worst at a rate of 59% at the high-frequency extreme. This is far greater than random (randomly selecting one of thirteen classes would be 7.69%). When the normalized confusion matrices given by figures 36, 37 and 38 are observed, it is noted that the performance of the

system degrades slightly with an increase in frequency. The reason for this is thought to be because of the use of the low-cost RTL-SDR, whose performance notably degrades with an increase in frequency. However, the author believes that machine learning techniques can overcome such hardware deficiencies if it *(i)* is fully trained and *(ii)* trained with more data.

The execution time of the system was measured as 0.186s on average. This is well within the specified 5s. The system does not use the Jetson Nano's GPU cores in the final implementation, as it is faster to use the CPU. The GPU acceleration was originally brought in to aid in the training process, where rather large matrices were multiplied at every step. However, in the AMC system implementation, this is not the case, the overhead of transferring data to and from the Jetson Nano's GPU nullifies the computational speedup obtained from using it.

5.2 Critical evaluation of the design

5.2.1 Aspects to be improved in the present design

The single greatest aspect to be improved in the present design is to change the implementation such that it is able to converge on a significantly better loss minimum. This could be achieved by using a larger training set or experimenting with different optimizer algorithms.

Furthermore, the system's performance below 20 dB SNR could be improved by incorporating dynamic gain adjustment as to always scale the incoming signal to the full dynamic range of the RTL-SDR.

This design is lacking in some forms of diversity in the training set. It does not incorporate many different symbol rates in digital modulation schemes. Generally in the literature the symbol rate is fixed, however, this should not be the case.

In this design, the RTL-SDR stands out as a tentative performance bottleneck. The RTL-SDR has a very low cost as SDRs go. The system as a whole would do better in every aspect if a slightly better low-cost SDR was used. That being said, notably, the system still functions despite using the RTL-SDR.

5.2.2 Strong points of the current design

The strongest asset of the present design is that its validation accuracy during training directly reflects its performance on real signals in the final system. This suggests that the exclusive use of real data for the training of an AMC system serves to build a better model that is robust in the real signal environment. This point is particularly emboldened by the 69% average classification accuracy obtained by the system using a model that was only half-trained.

The system also functions across different SDRs of the same model. The system was operated using two different RTL-SDRs which were not used to generate the training set. The system continued to work undeterred by this change of hardware.

5.2.3 Under which circumstances is the system expected to fail?

The system in its present condition is expected to fail if there is more than one signal in the band being considered. Any signal overlap in the band of interest will stop the system from reliably performing its function.

If the SNR of the signal of interest drops below 20 dB the system will begin to fail. Below 10 dB the system is expected to fail.

The system will fail if it encounters unknown modulation schemes it has not been prepared for. This is also true for other signal sources of known modulation as no tests were conducted to verify that the system works independently of the signal source. This is also true for other symbol rates.

5.3 Design ergonomics

Since this project is comprised entirely of software, no ergonomic considerations were able to be identified, hence none were considered.

5.4 Health, safety and environmental impact

The project is a software project. The software does not pose any health risk to the user. It is perfectly safe to use. No environmental impact of the AMC system could be identified as it does not emit any electromagnetic waves or introduce any substances of any kind into the external environment.

5.5 Social and legal impact of the design

This system could potentially be used for spectrum regulation. This could have legal impacts insofar as preventing unlawful use of spectrum.

6. Conclusion

6.1 Summary of the work completed

This document pertained to the work that was completed towards designing and implementing an AMC system, specifically using machine learning. A literature survey was done to familiarize the author with the history and state-of-the-art of AMC and machine learning techniques.

The Jetson Nano Dev Kit v3 was selected to run the AMC system, due to its onboard GPU that facilitates fast execution of the CNN. The RTL-SDR was chosen to be the SDR receiver for the system.

The AMC system was developed by designing and implementing a CNN. This CNN was then trained on a comprehensive dataset of 13 distinct modulation schemes with varying parameters, carrier frequency and SNR.

The dataset was developed using the HACK-RF One and GNU Radio to transmit signals over an RF coaxial cable that are subsequently received by the RTL-SDR and stored in a CSV file.

The performance of the system was evaluated experimentally to determine if the system was performing as expected.

6.2 Summary of the observations and findings

The system fell short of its core accuracy specification of 75% accuracy at 20 dB SNR by 6%, attaining an average accuracy of 69%. The system succeeded in identifying modulations and did so across frequency. The execution time of the system was measured to be significantly less than $5s$ at $0.186s$.

Overall, the proposed AMC system performed very well and holds much promise, despite training only halfway and falling short of the accuracy specification. The model consistently converged on a local minimum point on the surface of the loss function. As stated previously, this could be rectified by using more data, changing optimizer algorithms and further tuning hyperparameters.

6.3 Contribution

Before embarking on this project this student had no knowledge of RF or SDRs, as this is not included in the Computer Engineering undergraduate course work. The student, therefore, learned about RF and SDRs in theory and practice. Due to the scope of the project, the GNU Radio API was used to operate the SDRs; this was also new knowledge.

Convolutional neural networks and all mathematical components excluding the base artificial neural network concept was completely new to the student. Completely new code was

developed for CNNs with a strong reliance on libraries for core mathematical operations and transformations.

The student consulted with his study leader every week during this work. The study leader consistently provided useful general guidance; including providing the student with access to the CSIR's CHPC for use in his project. The study leader recommended the use of the HACK-RF One SDR for transmitting RF modulated signals. No other specific design or implementation decisions were recommended by the study leader; all other decisions were made at the student's discretion.

The previous student who completed this project was briefly consulted regarding tips to reduce the execution time of the CNN.

6.4 Future work

The foremost aspect that future work must look at is improving the execution time of the training routine. This can be accomplished by developing the CNN in a compiled language such as C or C++, still employing GPU acceleration. This will allow training on more data in less time.

Real data was used exclusively in this work, it is recommended that in future work a larger dataset of real data is used, in particular with more consideration of symbol rates. This aspect ties into the training time issue. because more samples require more time to train on. This author believes that improving upon the aforementioned two aspects will greatly increase the performance of the system.

Furthermore, this system did not consider batch normalization. This is a very useful machine learning technique that adds some complexity but completely makes up for it in delivering significantly faster model convergence. Hence any future work using CNNs for AMC should consider batch normalization.

There have been improvements in the methods for performing convolutions in a CNN as fast as possible. It is recommended that future work on some level considers the Winograd fast convolution, as it is the more efficient but still rather unpopular successor of Im2col.

7. References

[1] O. Dobre, A. Abdi, and W. Su, "Survey of automatic modulation classification techniques: Classical approaches and new trends," *IET Communications*, vol. 1, pp. 137 – 156, May 2007.

[2] F. Meng, 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. 10 760 – 10 772, Nov. 2018.

[3] B. Kim, J. Kim, H. Chae, D. Yoon, and J. W. Choi, "Deep neural network-based automatic modulation classification technique," in *International Conference on Information and Communication Technology Convergence*, Oct. 2016, pp. 579–582.

[4] N. Ghani and R. Lamontagne, "Neural networks applied to the classification of spectral features for automatic modulation recognition," in *Proceedings of MILCOM '93 - IEEE Military Communications Conference*, vol. 1, Oct. 1993, pp. 111–115 vol.1.

[5] S. Kremer and J. Shiels, "A testbed for automatic modulation recognition using artificial neural networks," in *CCECE '97. Canadian Conference on Electrical and Computer Engineering. Engineering Innovation: Voyage of Discovery. Conference Proceedings*, vol. 1, May 1997, pp. 67–70 vol.1.

[6] F. Rosenblatt, "Perceptron Simulation Experiments," *Proceedings of the IRE*, vol. 48, no. 3, pp. 301–309, Mar. 1960.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[8] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," *ArXiv*, vol. abs/1803.08375, 2018.

[9] C. Nwankpa, W. L. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *ArXiv*, vol. abs/1811.03378, 2018.

[10] J. S. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, ser. NIPS'89. Cambridge, MA, USA: MIT Press, Jan. 1989, p. 211–217.

[11] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," *ArXiv*, vol. abs/1805.07836, 2018.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[13] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, Dec. 2014.

[14] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, Sept. 2016.

[15] C. Cortes, M. Mohri, and A. Rostamizadeh, "L2 Regularization for Learning Kernels," in *UAI*, 2009.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, June 2014.

[17] T. O'Shea, J. Corgan, and T. Clancy, "Convolutional Radio Modulation Recognition Networks," in *Conference: International Conference on Engineering Applications of Neural Networks*, Feb. 2016.

[18] Y. Wang, J. Yang, M. Liu, and G. Gui, "LightAMC: Lightweight Automatic Modulation Classification via Deep Learning and Compressive Sensing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3491 – 3495, Feb. 2020.

[19] H. Zhang, M. Huang, J. Yang, and W. Sun, "A Data Preprocessing Method for Automatic Modulation Classification Based on CNN," *IEEE Communications Letters*, vol. 25, no. 4, pp. 1206 – 1210, Apr. 2021.

[20] D. Hong, Z. Zhang, and X. Xu, "Automatic modulation classification using recurrent neural networks," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, Dec. 2017, pp. 695–700.

[21] K. Liao, Y. Zhao, J. Gu, Y. Zhang, and Y. Zhong, "Sequential Convolutional Recurrent Neural Networks for Fast Automatic Modulation Classification," *IEEE Access*, vol. 9, pp. 27 182–27 188, Jan. 2021.

[22] B. P. Lathi, *Modern digital and analog communication systems*, 3rd ed., ser. Oxford series in electrical and computer engineering. Oxford University Press, 1998.

[23] S. Haykin and M. Moher, *Introduction to analog & digital communications*, 2nd ed. John Wiley & Sons, Inc., 2007.

[24] S. Russel and P. Norvig, *Artificial intelligence: A modern approach*, 3rd ed. Pearson, 2010.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," February 2015.

Part 4. Appendix: technical documentation

HARDWARE part of the project

Record 1. System block diagram

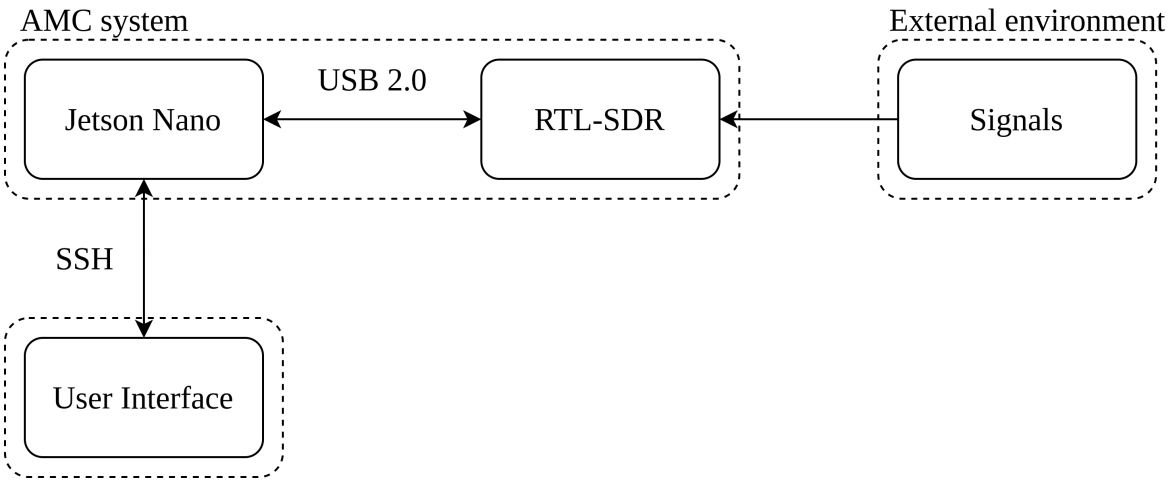


Figure 39.
System hardware block diagram

Record 2. Systems level description of the design

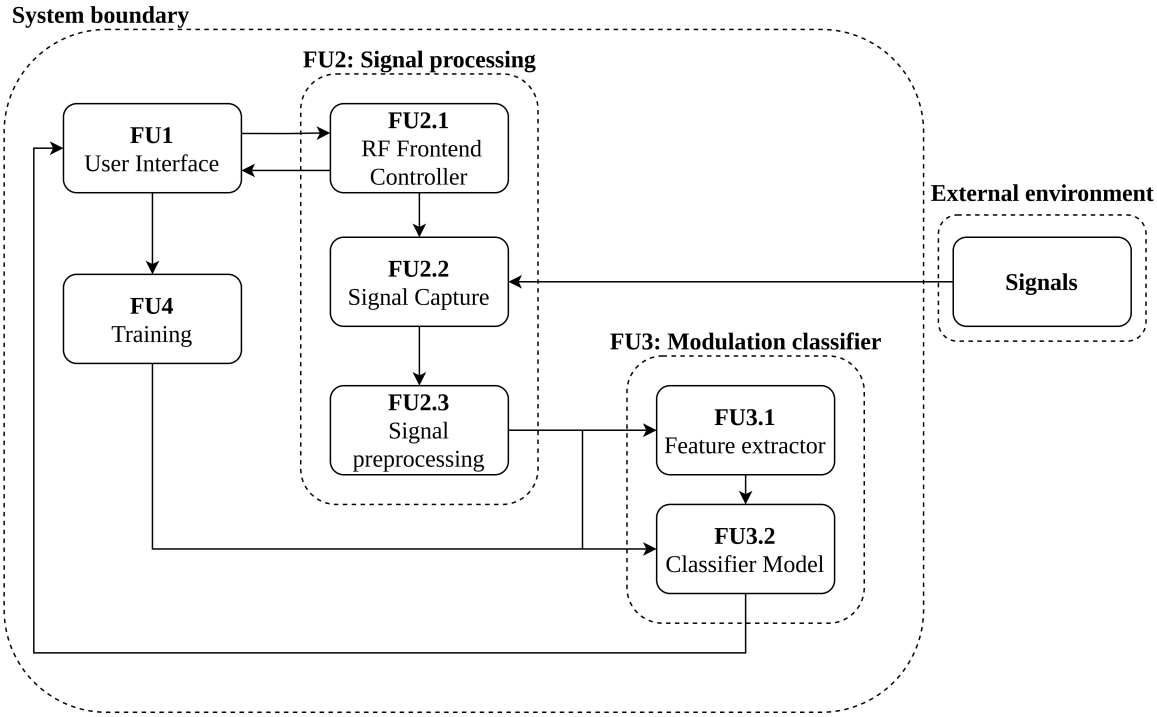


Figure 40.
System functional block diagram

Record 3. Complete circuit diagrams and description

Not applicable.

Record 4. Hardware acceptance test procedure

After the system has been powered on, the Jetson Nano’s green power LED marked "PWR" should be on and stable. This indicates that the Jetson Nano has power and is booting up.

The RTL-SDR is connected to the Jetson Nano correctly if the white LED on the RTL-SDR is on and stable.

Record 5. User guide

See Record 10, Software User Guide.

SOFTWARE part of the project

Record 6. Software process flow diagrams

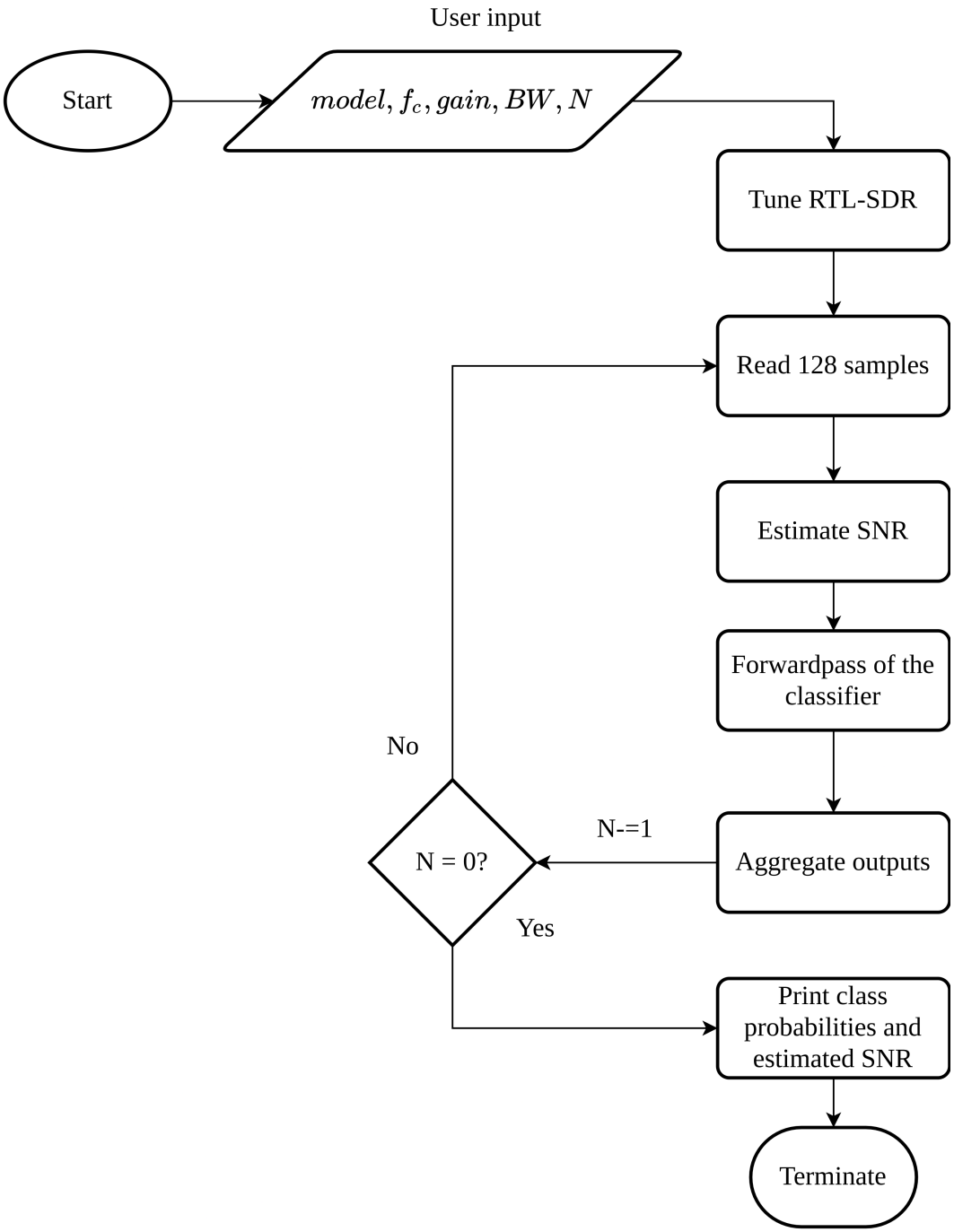


Figure 41.
AMC system software flowchart

Record 7. Explanation of software modules

A brief description is given of the utility of each of the functions in the AMC system program:

- **init()** - Instantiates the amcnet class, initializes the variables and loads the saved model with the same name as the input string.
- **matmul()** - Multiplies two input matrices and returns the resultant matrix. This function exists such that the system can still work even when no GPU is available.
- **loadNetwork()** - Load the weights, biases and filters stored in a *.npy* file.
- **saveNetwork()** - Save the weights, biases and filters to a *.npy* file.
- **forwardpass()** - Performs one forward pass of the whole structure and returns the resultant predicted class.
- **im2col()** - Executes the im2col transformation on the input.
- **col2im()** - Executes the col2im transformation on the input.
- **relu()** - Activates all input values using ReLU.
- **softmax()** - Calculates the softmax probability distribution of the input vector.
- **cross_entropy()** - Calculates the categorical cross-entropy loss for the input predicted class and actual class.
- **prepare_filters()** - Prepares the input filters for use in convolution by matrix multiplication by restructuring their form.
- **identifyModulations()** - Tunes the RTL-SDR to a specified carrier frequency and gain, then collects N samples, runs them through the classifier and returns the aggregated output.
- **adam()** - Computes the Adam optimizer algorithm.
- **dropout()** - Computes a dropout mask given the size and dropout rate of a dense layer.
- **loadDataset()** - Loads the specified dataset into memory.
- **backpropagate()** - Computes one full backpropagation of the structure given the input sample.
- **train()** - Trains the model for a specified number of epochs.

Some key functions and datatypes from the well-known Python library Numpy were used. The following list details the functions that were used and a brief description of their utility:

- **dot()** - Performs vector and matrix multiplication.

- **reshape()** - Transforms an *N* dimensional array into a different shape with the equivalent number of entries.
- **exp()** - Calculates the exponential function e^x.
- **sum()** - Calculates the sum over all the indices of the input.
- **array()** - Transforms other array datatypes to the Numpy array type.
- **expand_dims()** - Adds an empty dimension to an *N* dimensional array.
- **argmax()** - Returns the index of the maximum value found in the input.
- **zeros()** - Allocates memory for an *N* dimensional array of specified dimensions initialized with zeros.
- **ones()** - Allocates memory for an *N* dimensional array of specified dimensions initialized with ones.
- **random.randn()** - Draws a sample from a normal distribution with specified mean and variance.
- **float32** - Datatype for single precision floating point format.

The libraries pycuda and scikit-cuda were used to interface with GPU hardware using CUDA. This was implemented to outsource matrix multiplication to GPU hardware to speed up the execution of the program. The following details a list of functions that were used to enable this approach:

- **skcuda.linalg.dot()** Performs matrix multiplication of two matrices stored in GPU memory.
- **skcuda.gpuarray.to_gpu()** Writes array to GPU memory from main memory.

From the numexpr library the **numexpr.evaluate()** function was used in places to facilitate multiprocessing where possible. This library is called numpy express, because it sports parallel implementations of many numpy functions, including the evaluation of expressions.

Record 8. Complete source code

Complete code has been submitted separately on the AMS.

Record 9. Software acceptance test procedure

The readiness of the software can be tested by executing the following command in the terminal of the Jetson Nano:

```
$ rtl_test
```

This test for the RTL-SDR is passed if it returns that an RTL-SDR device has been found and is ready for operation. Subsequently when the command as described in the user guide is executed to run the program, the output should look like figure 42. If this is the case then the software acceptance test has been passed.

Record 10. Software user guide

This software automatically identifies modulation schemes on a signal channel with no information other than the signal itself. The model can identify thirteen different RF modulation schemes. These categories are represented in table 10.

AM	FM	PSK	QAM	FSK
AMDSB	WBFM	2PSK	8QAM	2FSK
AMUSB-SC		4PSK	16QAM	GFSK
AMLSB-SC		8PSK	32QAM	GMSK

Table 10.
Modulation schemes

The system works best when used at a signal to noise ratio of 20 dB (as is indicated by the software), at a lower frequency relative to the entire range of RTL-SDR. It is recommended to use the system at the full RTL-SDR bandwidth of 2.4 MHz.

Before operating the system, the hardware acceptance tests presented in Record 4 should be followed.

Once an SSH terminal to the Jetson Nano has been established, to run the AMC system, the following command should be used:

```
$ python3 amc.py modelname f_c gain bw N
```

This command executes the AMC system script with a number of command-line parameters that influence the operation of the system. The *modelname* parameter represents the file of weights, biases and filters that is loaded. The following parameters represent the centre frequency, gain, bandwidth and number of measurements that the RTL-SDR will be configured to in the program run. The system should be operated at the full bandwidth of 2.4 MHz.

The system will safely cease operation if the user enters CTRL + C into the SSH terminal.

The standard way to run the system is with the following command:

```
$ python3 amc.py amcnet 90e6 0 2.4e6 4
```

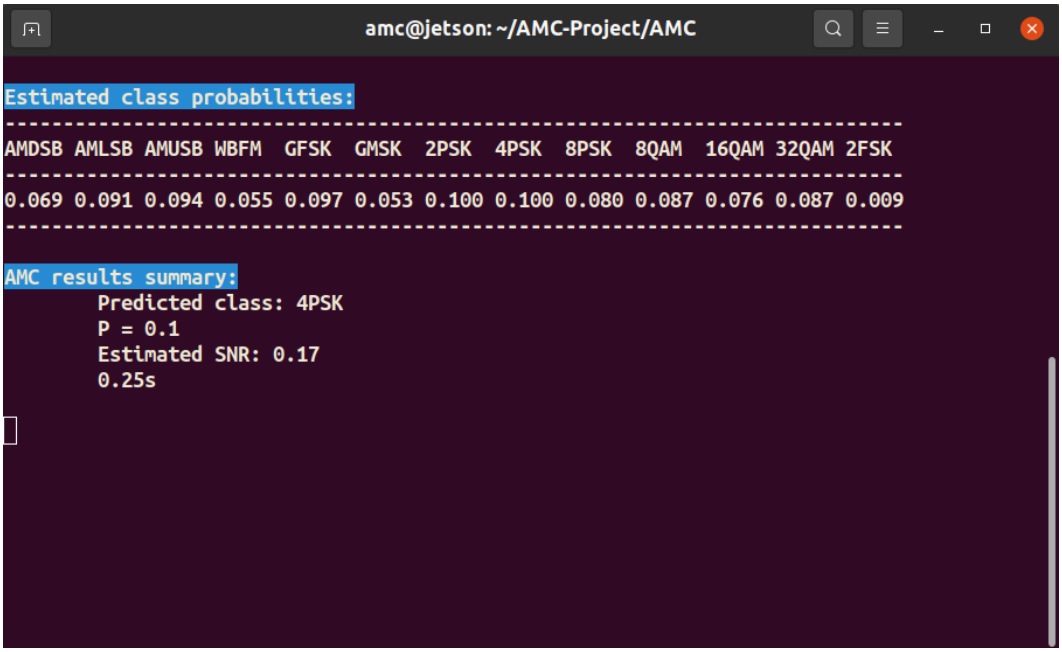


Figure 42.
AMC system software screenshot

EXPERIMENTAL DATA

Record 11. Experimental data

The following confusion matrices detail the systems performance at different SNR levels.

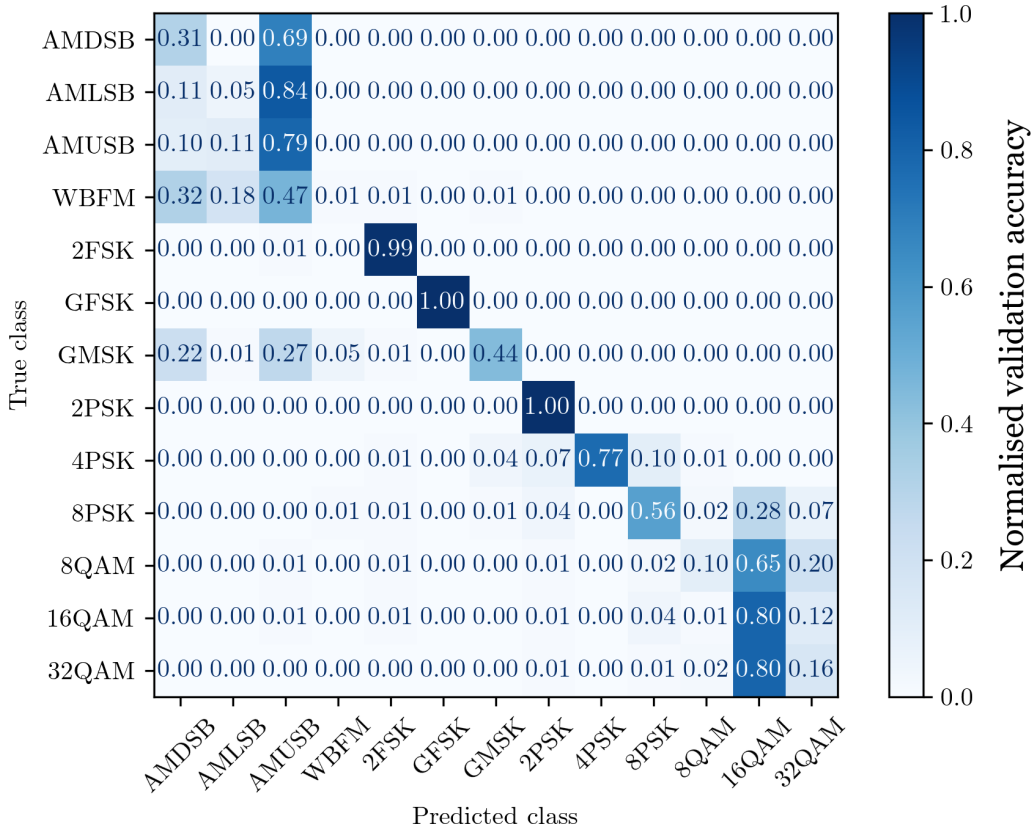


Figure 43.
Normalized confusion matrix of AMC system at 10 dB

The average performance is obtained by taking the mean of the diagonal, resulting in 53%.

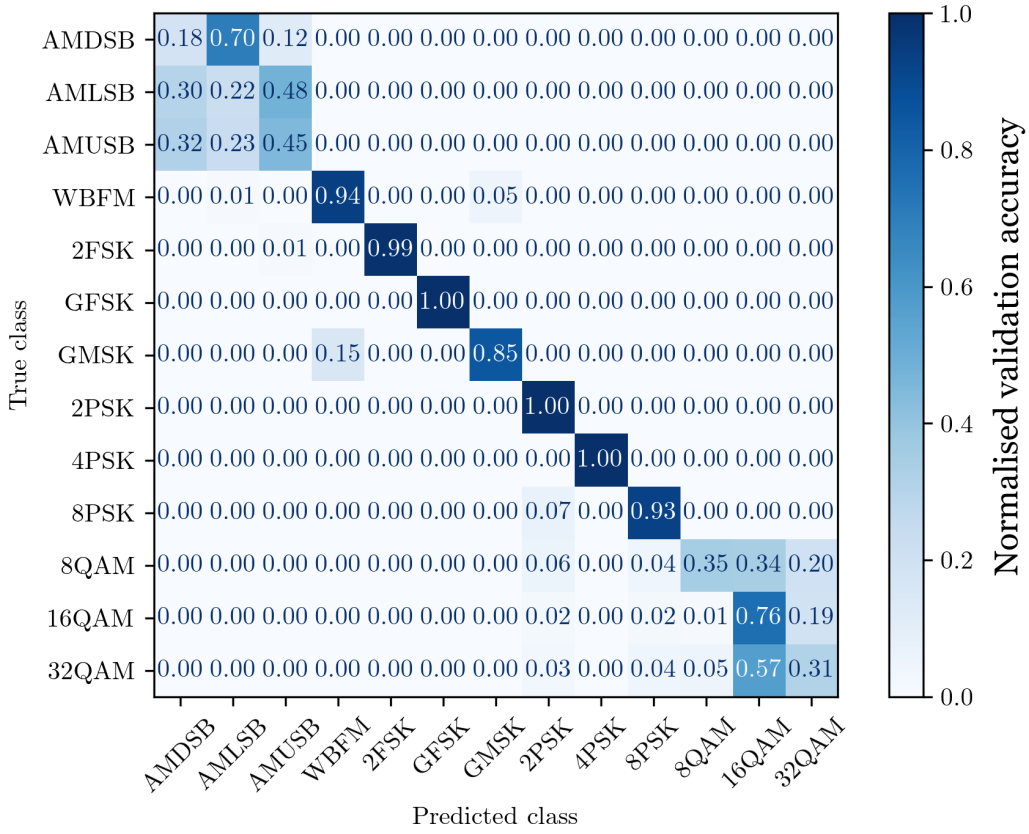


Figure 44.
Normalized confusion matrix of AMC system at 20 dB

The average performance is obtained by taking the mean of the diagonal, resulting in 69%.

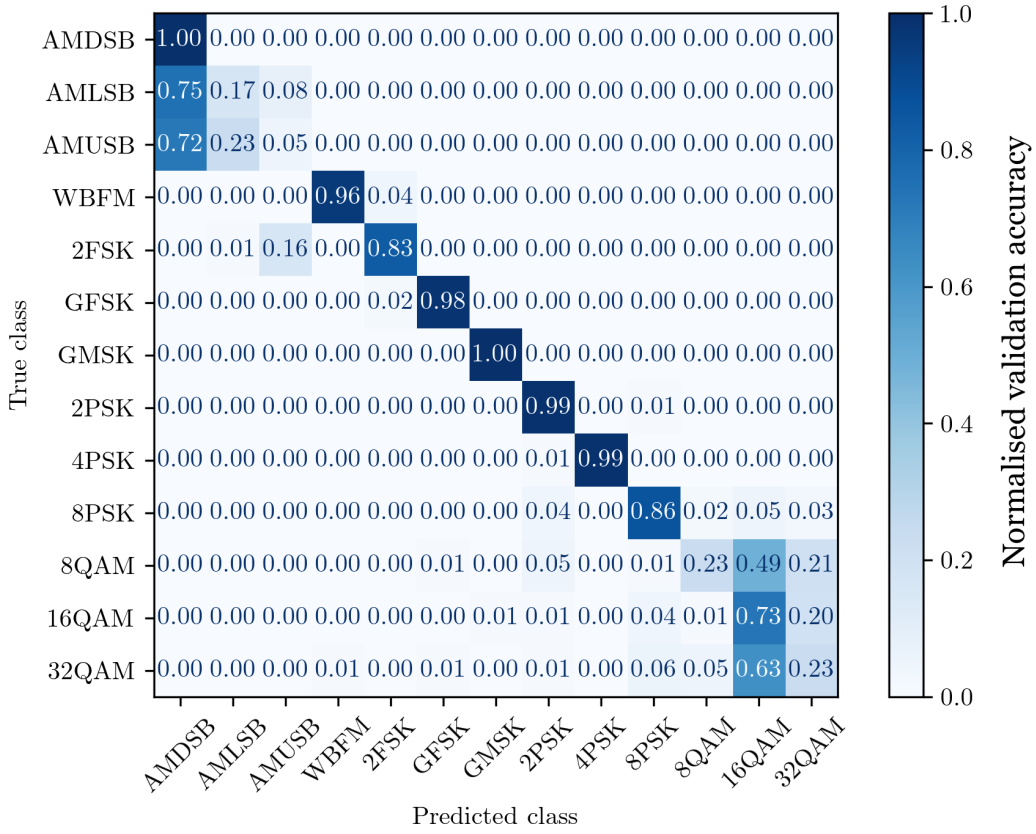


Figure 45.
Normalized confusion matrix of AMC system at 30 dB

The average performance is obtained by taking the mean of the diagonal, resulting in 69%.