BOWIE STATE UNIVERSITY

ONE-SHOT ADAPTIVE CLASSIFICATION IN DYNAMIC ENVIRONMENTS: A STUDY USING RADIO FREQUENCY WAVEFORMS

A Dissertation Submitted to the Faculty of the Graduate School

BOWIE STATE UNIVERSITY

In Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF SCIENCE

Department of Computer Science

By

Marvin A. Conn, Sr.

November 5, 2020

BOWIE STATE UNIVERSITY

THE GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE

DISSERTATION COMMITTEE:

DocuSigned by:
Dr. Darsana Josyula
42339CA0E58C493...
Dr. Darsana Josyula, Ph.D.
Chair

DocuSigned by:
4A098D1FA377418...
Dr. Manohar Mareboyana, Ph.D.

DocuSigned by:
Dr. Soo-Yeon Ji
25217B3C10D7489...
Dr. Soo-Yeon Ji, Ph.D.

DocuSigned by:
A045319A0FA047A...
Dr. Gil F. de Lamadrid, Ph.D.

DocuSigned by:
Dr. Anthony Martone
0234D91E1E04476...
Dr. Anthony Martone, Ph.D.
External Examiner
Army Research Laboratory
Powder Mill Road, Adelphi, MD 20783

Candidate: Marvin A. Conn, Sr.

Date of Defense: November 5, 2020

ABSTRACT

Title of Dissertation: ONE-SHOT **ADAPTIVE CLASSIFICATION IN DYNAMIC ENVIRONMENTS: A STUDY USING RADIO FREQUENCY WAVEFORMS**

Marvin A. Conn, Sr.

Dissertation Chaired by: Dr. Darsana P. Josyula, PhD
Department of Computer Science
Bowie State University

The radio frequency (RF) spectrum is a critical and limited resource for communication and sensing. Due to technological advances, the use of RF devices is increasing exponentially leading to RF crowding, and therefore presenting significant challenges in avoiding interference and improving spectrum sharing. A general solution to the problem lies in developing cognitive RF architectures, which allow for the autonomous operation of RF systems using artificial intelligence techniques. Significant research suggests that artificial intelligence is necessary for cognitive architectures to reason and adapt in dynamic environments.

This research addresses waveform classification challenges of cognitive RF architectures that first uses supervised learning techniques for waveform classification of a set of known waveforms, and when taken online must continue to classify the known waveforms as well as to on-the-fly recognize and learn to classify waveforms never seen before. Online in this research is defined as scenarios having streaming discrete or continuous data inputs that continuously arrives, rather than in batches, and therefore requires incremental processing to transform the data into

useful information for sensor-based measurements and real-time decision-making. This study explores the effectiveness of combining supervised transfer learning, and unsupervised one-shot learning to support online adaptive classification algorithms. We begin by using supervised transfer learning with CNNs to train classifiers on known waveform constellation image datasets. We then explore using the trained CNN architectures to generate useful feature vectors as inputs into our adaptive classification algorithms. Although the algorithms developed were using RF waveform datasets, they can be used in any domain providing representative feature vectors for the classes.

The contribution of this research led to the development of four novel adaptive algorithms for classification, anomaly detection, and clustering of RF waveform data. The first two algorithms using supervised and one shot learning centroid models are the Centroid with Anomaly Detection and Clustering (CADC) algorithm, and the Frequency Hits Anomaly Detection and Clustering (FHITS) algorithm. To address a limitation of DYNG [1] algorithm, in which it does not perform anomaly detection for the real-time labeling of anomalous data input to the network, we create two additional algorithms by using CADC and FHITS as strategies for the detection and insertion of anomalous classes into the DYNG network. These two algorithms are called DYNG-CADC and DYNG-FHITS and allow for DYNG to perform real-time labeling of anomalous stimuli belonging to different classes.

In this research we performed a comparative evaluation of CADC, FHITS, DYNG-CADC, and DYNG-FHITS to determine how well they perform when presented with anomalous stimuli. As part of our research we used static CNN architectures as feature generators for inputs to our algorithms, and we effectively demonstrated the CNNs can be transformed into adaptive classifiers by replacing the last layer with our adaptive algorithms. Our work also explored principal

component analyses (PCA) to devise a highly accurate anomaly detector applied to CNNs. Finally, we devised a Random M-class anomaly detector allowing for M unknown classes to be detected within CNNs, however we concluded such an algorithm is not practical for online adaptive systems without further modification.

ACKNOWLEDGEMENTS

As I celebrate this journey, To God Be the Glory, for giving me the faith, courage, and strength in remaining steadfast in completing this most challenging endeavor. No words can express my heartfelt gratitude towards my family, friends and colleagues that have shown me support as I pursued this dream. I thank all my committee members for your invaluable feedback, and for challenging me with critical questions to be addressed during my research. I express my deepest appreciation to my committee chair, Dr. Darsana P. Josyula, for your unwavering encouragement, compassion, patience, and support. I thank you for being a true educator and for providing me with invaluable guidance towards completing my research. I would thank Dr. Oliver Beyer for providing source code for the DYNG network, as it proved to be an invaluable resource for my research. I thank my dear parents, Willard Conn, Thelma Black, and Willie Black, for your unwavering encouragement and frequent pushes for me to not rest until my research was done. I thank my children Milaina, Caitlin, and Marc for your encouragement, and it is because of your eyes being laid upon me that I could not quit. I had to practice what I often preach to you, that one should never give up when presented with challenges in life. I thank my brothers William Conn and Quentin Jones for your encouragement and occasional check-ins to inquire how my doctoral studies were going. I thank the Army Research Laboratory for its tremendous support while I pursued my doctoral degree. Thank you to ARL colleagues Dr. Paul Amirtharaj, Dr. Romeo Del Rosario, Mr. Eric Adler, Mr. Kwok Tom, Mr. Derwin Washington, and Dr. Anthony Martone. Finally, I thank my loving and dear wife, Dr. Debbie Black-Conn, for your patience, guidance, encouragement, and support, particularly during my most difficult times. If it were not for your love and encouragement, I would not have been successful on this monumental journey.

TABLE OF CONTENTS

LIST OF ACRONYMS

16QAM	16 Quadrature Amplitude Modulation
32QAM	32 Quadrature Amplitude Modulation
4ASK	Quadrature Amplitude Shift Keying
8PSK	8 Phase Shift Keying
ACC	Overall Accuracy
ADC	Analog to Digital Converter
AE	Auto Encoder
AFRL	Air Force Research Laboratory
AI	Artificial Intelligence
AM	Amplitude Modulation
AM-DSB	Amplitude-Modulation Double-Sideband
ANN	Artificial Neural Network
ART	Artificial Dataset
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BFSK	Binary Frequency Shift Keying
BNP	Bayesian Nonparametric Model
BPSK	Binary Phase Shift Keying
CADC	Centroid with Anomaly Detection and Clustering
CNN	Convolutional Neural Network
CPFSK	Continuous-phase frequency-shift-keying
DAC	Digital to Analog Converter
DANN	Deep Associative Neural Networks
DARPA	The Defense Advanced Research Projects Agency
DDC	Digital Down Conversion
DEC	Dynamic Ensemble Classifiers
DNN	Deep neural networks
DPMM	Dirichlet Process Mixture Model
DSP	Digital Signal Processing
DUC	Digital Up-Converter
DYNG	Dynamic Online Growing Neural Gas
DYNG-CADC	DYNG augmented with CADC
DYNG-FHITS	DYNG augmented with FHITS
EDDB	Environmental Dynamic Database
FCC	Federal Communications Commission
FHITS	Frequency Hits Anomaly Detection and Clustering
FN	False Negative
FNR	False Negative Rate
FP	False Positives
FPGA	Field Programmable Gate Array

FPR	False Positive Rate
GAN	Generative Adversarial Network
GFSK	Gaussian Frequency Shift Keying
GMM	Gaussian Mixture Model
GMTI	Ground Moving Target Indicator
GNG	Growing Neural Gas
HOS	Higher Order Statistics
IQ	In-phase and Quadrature
ITU	International Telecommunications Union
KA	Knowledge Aided
LO	Local Oscillator
LPF	Lowpass Filter
MAD	Median Absolute Deviation
ML	Machine Learning
NAPR	Non-adaptive Prediction Rate
NG	Neural Gas
NTIA	National Telecommunications and Information Administration
OGNG	Online Growing Neural Gas
OQPSK	Orthogonal Quadrature Phase Shift Keying
OSSGNG	Online Semi-supervised Growing Neural Gas
PAM4	Four-Level Pulse-Amplitude Modulation
PCA	Principal Component Analyses
PLL	Phase Locked loop
PPV	Positive Predictive Value
PVAR	Percentage of PCA Variance
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RF	radio frequency
RFI	Radio Frequency Interference
RGB	Red-Green-Blue
SDR	Software Defined Radio
SNR	Signal to Noise Ratio
SOM	Self-Organizing Maps
SSB	Single Sideband
SVM	Support Vector Machine
TN	True Negative
TNR	True Negative Rate
TP	True positive
TPR	True positive rate
VAE	Variational Autoencoder
VCO	Voltage Controlled Oscillator
WBFM	Wideband Frequency Modulation

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

The human brain [2] is pre-wired with cognitive abilities to carry out visual tasks, for example the brain of newborns come with connections that precede the development of domain-specific functions. Much of the structure of a baby's vision system along with patterns of brain activity are already in place at birth, and the brain's facial and place recognition networks are connected and communicating with each other within days of birth. As the baby's brain matures over time, it evolves to contain a mosaic of domain-specific networks, allowing for the recognition of the different people and places it is exposed to.

From a classification perspective, when we are presented with objects and their associated labels, we memorize their representations. At future time steps when we are presented with that same or a similar object without a label, we can recall from memory our representation of the object and its associated label, and finally we can classify the presented object as the recalled label. In being armed with pre-wired and training knowledge, we operate autonomously in the environment, with no external teacher to provide the answers, and we fair quite well labeling new objects until presented with objects we have never seen the likes of before. In this scenario, we arrive at an impasse and we may resolve it one of two ways. The undesirable way is to declare the object as one of the objects we have trained on even though it has no resemblance, and this will only lead to performance degradation. The second and more desirable approach is to decide in an unsupervised manner if the object is a never before seen, and if true we assign the object a new label, and we save in memory relevant features about the object for future reference. In using this second approach, we reduce our chances of incorrectly classifying objects our teachers have taught us, and we teach ourselves to recognize new objects as they are presented to us in life.

The classification requirements of an adaptive cognitive radio frequency (RF) system can be likened to the development of a baby's visual recognition networks, or the classification perspective given above. Imagine such a cognitive RF system operating in a dynamic RF environment where it must recognize not only the known waveforms it has been trained on but also must evolve over time and learn to recognize new waveforms that it has never been trained on. To obtain this level of adaptive recognition, we first rely on prewired abilities by leveraging the initial knowledge obtained from supervised learning techniques.

Sharing of the RF bands has attracted substantial attention, to avoid under-utilization of permanently allocated spectral resources, and to avoid spectrum interference due to the increasing presence of RF systems that must share congested spectral resources. To improve efficiency of coexistence, there is a need to develop dynamic spectrum access capability using machine learning (ML) approaches, for signal classification to distinguish between signals from multiple users in the presence of noise and interference [3].

This research will use ML concepts to explore online RF waveform: classification; anomaly detection; and adaptation while operating in non-stationary RF environments. Online in this research is defined as scenarios having streaming discrete or continuous data inputs that arrives continuously, rather than in batches, and therefore requires incremental processing to transform the data into useful information for sensor-based measurements and real-time decision making. The general operating premise is that the classifiers are initially trained in a supervised learning mode on the known classes. Following the training procedure, the algorithms are taken online to accurately classify the known classes, and perform anomaly detection, clustering, and classification of anomalous waveforms classes, in a real-time fashion without being taken offline.

1.1 BACKGROUND

RF spectrum is defined as the electromagnetic waves in the 3 kHz to 300 GHz frequency range and is used by wireless devices such as radios and radars facing increasing challenges in gaining access to this limited and critical resource. One such challenge is RF interference (RFI) caused by other RF sources in or near the operating bands, for example it has been shown that RFI significantly degrades performance for air traffic control radars and weather systems [4].

RF spectrum access is managed by government organizations to prevent interference by assigning licenses to users operating in assigned RF bands. Within the United States, the Federal Communications Commission (FCC) manages non-federal users, and the National Telecommunications and Information Administration (NTIA) manages federal users. Globally, RF spectrum is managed by the International Telecommunications Union (ITU) for coordination among countries where each country has membership organizations similar to the FCC and NTIA advocating for their individual interests [5], [6].

Device access to the RF spectrum can be cooperative or non-cooperative [7], [8], where cooperative communication is a method where users transmit and receive data using agreed upon communication protocols to optimize communication efficiency, whereas non-cooperative communication is considered a traditional multiple user access channel, where users send directly to a common destination, without coordinating synchronization or repeating data for each other. It has been shown that when devices communicate in a cooperative manner the result is a significant improvement in error and transmission rate performance.

This research focuses on developing online adaptive RF waveform classification algorithms to provide RF systems with the ability to acquire knowledge for decision making

processes in the RF environment, and it supports both the cooperative and non-cooperative modes of operation. For example, in cooperative mode, it may be desirable to identify a specific waveform emitter type before communications are initiated. In the non-cooperative mode, it may be desirable to recognize and track the behavior of an adversarial class of RF waveforms that do not operate within any mutually agreed upon communications protocols between RF systems.

1.2 STATEMENT OF THE PROBLEM

The internet-of-things (IoT) are high bandwidth wireless devices driving the demand for access to RF networks to communicate and process data. IoTs are primarily used in the consumer, business, and government sectors, with examples being cellphones, cameras, and vehicles. The use of IoTs is growing exponentially with expert projections of 20-50 billion devices in use by the year 2025 [9]. Also driving the demand for spectrum access is the internet-of-battle-things (IoBT) which are wireless devices that communicate and process data for military applications. Such devices may include sensors, munitions, weapons, vehicles, robots, drones, communications, and human wearables. Applications may include emitter detection, geolocation, identification, and signal intelligence systems. The battlefield of the future will be densely populated with IoBTs where some are intelligent and others perhaps only marginally so [10], [11].

Although there is increased demand for spectrum usage and primary user allocation, there still exists the problem of spectrum underutilization. Many devices are designed to operate in exclusively licensed frequency bands with the assumption that the assigned spectrum is always available to that licensed user and will not be interfered with [12], [13], however the FCC Spectrum-Policy Task Force [14], standing guidelines prevent candidate spectrum users from accessing the unused licensed bands and therefore causing spectrum holes. Spectrum holes [15] implies that when

the spectrum is not in use it will remain available to the licensed user when needed in the future. However, this is inefficient because it does not allow non-licensed users to operate on the band when not used by the license holder.

The explosion of the use of wireless devices has led stakeholders and the FCC to indicate that the traditional approach of managing interference is becoming more challenging [9]. This has spurred tremendous research into discovering alternative techniques to cope with the problem. For example, light-based systems such as Li-Fi to implement Wi-Fi in localized areas or the use of infrared Lidar for wetland mappings have been investigated, however these approaches do not address the long-range requirements of wireless systems. Other research avenues are exploring the use of cooperative joint RF sensing and communications techniques to mitigate interference by using system architectures that for example can function not strictly as radar or communication devices, but as universal cognitive transceiver platforms that function both as radar and or communication systems by dynamically reconfiguring themselves to meet the immediate functional demands [12]. The Defense Advanced Research Projects Agency (DARPA) has funded a program to develop software defined radio (SDR) systems that autonomously collaborate, called "Collaborative Intelligent Networks". This research attempts to address the challenges of spectral coexistence. In 2017, DARPA launched a program called the Spectrum Collaboration Challenge (SC2) to encourage the development of radio networks that can autonomously collaborate to operate without the use of traditional spectrum allocation. As part of this effort, DARPA created a testbed called the "Colosseum" to emulate tens of thousands of possible interactions between hundreds of wireless communication devices in real time [16].

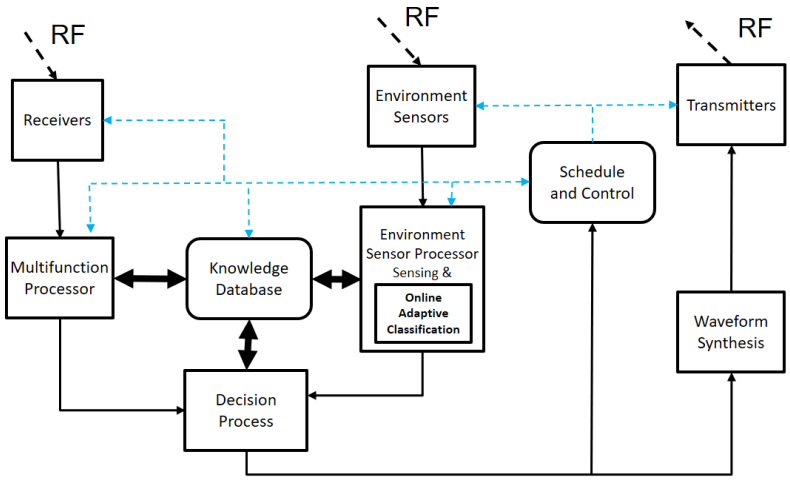


Figure 1. Generalized Cognitive RF Transceiver with Adaptive Classification [17]

Cognitive RF architectures such as shown in Figure 1, are used to carry out such research. The diagram of a Generalized Cognitive RF Transceiver is derived from the works of [17] with slight modification to depict a cognitive wireless transceiver device requiring adaptive classification capability which will be the focus of this research. The main components of the generalized cognitive RF architecture are multiple receivers, multiple environment sensors, multiple transmitters, a multifunction processor, a knowledge database, an environment sensor processor with adaptive classification, a waveform synthesizer, and a central scheduler and controller. Viewing Figure 1 in the context of radar system, the RF receivers receive radar return signals emitted from the RF transmitters, and in the context of radio communication the RF receivers receive signals emitted from remote radio transmitters. The blue lines indicate system timing and control flow. The bold lines indicate a bidirectional flow of information regarding radar targets and the environment. The plain lines indicate the processing flow for the RF transmitter, receiver, and environmental signals. Within the environment sensor processor is shown the online adaptive processing box which is where the work in this dissertation is focused. It is through the conceptualization of such generalized cognitive RF architectures that researchers are working towards reconciling the difficulties of RF spectrum usage and overcrowding.

An important aspect of cognitive RF architectures is having the ability to accurately classify multiple types of known waveforms acquired by its sensors. Such classifiers are typically trained offline with supervised learning techniques with a fixed number of classes [18-22] . In non-stationary RF environments, such classifiers will inevitably be presented with unknown waveform classes, different from what they were trained on (anomalies), and under such conditions they will degrade in performance due to misclassifications.[1]. Therefore, these systems must possess the real-time ability to detect anomalous waveforms and quickly learn their attributes so that RF classification can proceed without hindrance as new classes are introduced. These capabilities, among others, are a necessary lower-level component in the higher-level decision-making processes for cognitive RF transceivers and they interact with the Decision Process and Knowledge Database as indicated in the model of the Generalized Transceiver shown in Figure 1.

For classification tasks, supervised/semi-supervised learning techniques [23] have recently seen tremendous success with the advent of faster computers with massive storage capabilities and that can process vast amounts of data quickly. Training such classifiers relies on prior knowledge of the number of classes and labeled training data for classes. While online in non-stationary RF environments, the number of classes may increase over time, for example a waveform classifier trained to classify N types of waveforms must classify N+1 types of signals when a new unknown signal class (for example, 5G) is introduced into the environment. When the unknown class is presented to the classification task, a supervised learned classifier will misclassify all data belonging to that new class since the classifier does not have a mechanism to detect the unknown. This scenario leads to misclassifications that go unnoticed and degrades performance, therefore, the classifier system must detect when new classes are presented and must dynamically adapt to the unknown classes while maintaining overall performance.

[1] The performance degradation was empirically evaluated as part of this research.

A straightforward (but impractical) strategy to adjust the classifier is to retrain with labeled data of the new class along with the old training data. This strategy is not practical for several reasons: (1) a deployed classifier may not have access to the old training data, (2) labeled data for the new class may not be immediately available for re-training, and (3) such training will take at least the amount of time spent training with the original classes, and finally (4) the classifier has no mechanism in place to know when it is presented with a new class.

1.3 PURPOSE OF STUDY

The contribution of this research is to develop and assess adaptive classification algorithms suitable for the higher-level decision-making processes required for cognitive RF transceivers as shown in ***Figure 1***. The emphasis is on the exploration of components that are intended to reside within the sensor processor, knowledge database, and decision-making processes.

The scope of this research is broad and will extend into post-doctoral studies. The focus of this dissertation is in establishing algorithms to support waveform classification, anomaly detection, and online clustering techniques suitable for cognitive RF transceiver design in non-stationary environments.

This study addresses the following research questions:

- How do current ML techniques compare when transfer learning is used to classify known RF waveforms?
- What is the performance of these techniques in the presence of anomalous new RF waveforms?
- How can pre-trained classifiers detect anomalous new classes while online?

- How can pre-trained classifiers learn to adapt to new RF waveforms while online?
- What is the performance of the adaptive algorithms developed in this research?
- What is the computational complexity of the developed algorithms?

The results of the dissertation include:

- Description of the datasets and experiments used to carry out research
- Presentation of approaches and pseudocode for adapting pre-trained neural-network classifiers to detect and classify new classes while online
- Complexity analysis of the algorithms developed for adaptive classification
- Performance results of the algorithms and the resulting adaptive classifiers on simulated RF datasets

1.4 SIGNIFICANCE OF STUDY

The RF spectrum is becoming increasingly crowded, and a cognitive RF transceiver is perceived as a tremendously important solution to obtain the most efficient use of the spectrum. A cognitive RF system must be fully aware of its environment to reliably make proper decisions in navigating use of the spectrum. Full automation without human intervention is required due to tight timing constraints of RF systems. Cognitive RF systems must be capable of assessing conditions under changing RF environments and must be able to make real time decisions that do not hinder performance. The cognitive RF transceiver architecture implies a need for acquiring real time knowledge of the characteristics of RF emitters present in the operating environment. Equipping a cognitive RF system with such capabilities should only work to enhance performance.

This study contributes to the area of cognitive RF systems particularly by providing a methodology for online unsupervised adaptation of pre-trained classifiers to deal with new RF waveforms. RF spectrum classification and adaptive anomaly detection algorithms are critical components of cognitive RF systems. Without such components, the cognitive RF architecture will always be limited in terms of the granularity in which their decision-making processes can take place. The more information that can be determined about an unknown or anomalous waveform emitter, the more decision-making power can be afforded to a cognitive RF system.

The findings of this study support the implementation of cognitive RF systems by providing novel techniques for adaptive RF spectrum classification, and for unsupervised learning and classification of anomalous waveforms. The study also validates the use of transfer learning to

generate feature sets for RF waveform data and using the resultant feature sets as inputs to adaptive classifiers.

The significance of this study is not only limited to the RF spectrum domain because the algorithms developed can be used in any dynamically changing data domain requiring online real-time adaptive classifiers. The relevance of this research extends to a much broader range of applications such as intrusion detection, medical diagnosis, and fault detection systems.

2 LITERATURE REVIEW

This dissertation work draws from previous work in several different research areas including modulation classification, anomaly detection and clustering. This chapter provides a survey of literature related to the automatic one-shot adaptation of classifiers developed in this work to deal with new classes of data. It provides an overview of relevant definitions including different distance measures used to determine the similarity or dissimilarity between two instances, followed by overview discussions on classification, clustering and anomaly detections techniques that are commonly applied in the literature. It follows this with a discussion on cognitive radars and cognitive radios and how RF modulation classifiers fit within the overall framework of cognitive radars and cognitive radios.

2.1 DEFINITIONS

This section provides fundamental definitions of some of the key concepts referred to throughout this dissertation.

2.1.1 Online Streaming Data Processing

Streaming data, where the raw data can have either discrete or continuous values, arrives continuously rather than in batches and requires incremental processing where one can transform the raw data into useful information for sensor-based measurements[24], for example, users may be concerned about real-time results such as detecting anomalies within a specific time period [25]. Online streaming Big data is defined as datasets whose size is beyond the capacities of typical software and hardware architectures to fully capture, store, and process, due to timing and memory constraints [26]. Therefore, online Big data stream processing techniques cannot store the entire

raw streaming data to capture complete information about the data, therefore there is an interest in using online learners that constantly update their structure while processing the incoming instances of associated classes to perform classification [27].

The algorithms developed in this research are suitable for online learning and classification in streaming Big data scenarios.

2.1.2 Distance Measures

Several distance algorithms are used as the basis for classification, clustering, and anomaly or change detection [28], for example, detailed performance tests using the Euclidian, Manhattan, Lance and Williams, Quadratic, Bhattacharya, Kullback-Leibler, and Delta algorithms have been applied to change detection on acoustic data [29]. This section provides a brief overview of some commonly used distance measures that are crucial for developing detection and classification algorithms.

2.1.2.1 Euclidean

The most widely used distance measurement is the Euclidean distance. The Euclidean distance, also referred to as the L2 norm, is the most preferred method because it is preserved under orthogonal transformations such as the Fourier transform [30]. Given two n dimensional vectors p and q, where $p = (p_1, p_2, \dots p_n)$ and $q = (q_1, q_2, \dots q_n)$, the Euclidean Distance between them is defined as $D(p, q) = \sum_{i=1}^{n} \sqrt{(p_i - q_i)^2}$.

2.1.2.2 Manhattan

The definition of the Euclidean distance is like that of the Manhattan distance. The primary difference is that the Manhattan distance uses absolute differences instead of squared differences and is more computationally efficient. The Manhattan distance, also referred to the L1 norm [31], between two vectors p and q is defined as

$$D(p,q) = \sum_{i=1}^{n} |p_i - q_i| \, .$$

2.1.2.3 Minkowski

The distance from a query point x_q to an example point x_j is typically measured with the Minkowski distance defined as

$$L^p(x_j, x_q) = \left(\sum_i |x_{j,i} - x_{q,i}|^p\right)^{1/p}.$$

When $p = 2$ the result is the Euclidean distance, and with $p = 1$ the Manhattan distance. Often $p = 2$ is used when the dimensions are measuring similar properties such as length, width, and height. p is typically set to 1 if measuring distances between dissimilar properties such as red, width, and velocity [32].

2.1.2.4 Chebyshev

The Chebyshev distance [33] between two vectors p and q for all i is defined as

$$D(p,q) = \max_i |p_i - q_i| \, .$$

2.1.2.5 Mahalanobis

Given a normal probability distribution $N(\mu, \Sigma)$, x is a d-component column vector, μ is a d-component mean vector, Σ is the d-by-d covariance matrix, and Σ^{-1} is the inverse, the Mahalanobis distance provides a separation measurement between the vector x and μ is defined [31] as

$$D(x) = (x - \mu)^T \Sigma^{-1} (x - \mu).$$

2.1.2.6 Bhattacharya Coefficient

The Bhattacharyya coefficient is a measurement of the amount of overlap between two statistical samples[34]. The coefficient can be used to determine the relative distance between the two samples under consideration. The interval of the values of the two populations is split into several partitions, and the number of members in each partition is used in the formula

$$C(p, q) = \sum_{i=1}^{n} \sqrt{p_i q_i}\ .$$

The sample populations are p and q, n is the number of partitions, and p_i and $q_i$are the number of member samples of p and q in the i-th partition.

2.1.2.7 Bhattacharya Distance

The Bhattacharyya distance [34] measures the similarity of the two probability distributions $p(x)$ and $q(x)$. It is closely related to the Bhattacharyya coefficient. For discrete probability distributions p and q over the domain x, the Bhattacharyya distance is defined as

$$D_B(p, q) = -\ln\left(BC(p, q)\right)$$

where

$$BC(p,q) = \sum_{x \in X} \sqrt{p(x)q(x)}\,.$$

The measure can be used to determine the separation of the two samples under consideration and is considered a more reliable measure than the Mahalanobis distance.

2.1.2.8 Mean Squared Error

A common cost function for training neural networks, comparing last layer output to the target training sets [31] is the mean squared error defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2,$$

where n is the number of Y_i observations and $\widehat{Y}_i$ are the predictions from an estimating model.

2.1.2.9 Entropy

The concept of entropy was developed by Claude Shannon to establish mathematical theories of information communications. The fundamental idea is to reliably and efficiently transmit a message from sender to receiver. Entropy measures the amount of uncertainty of the outcome of a random variable[35].

Entropy can also be used to understand the theoretical limits of lossless data compression by providing the mathematical framework to define how few bits are needed to represent the information of a signal source without losing information. A random event E with true probability $P(E)$ is said to contain $I(E) = -log_2 P(E)$ bits of information. If $P(E) = 1$, then $I(E) = 0$ interpreted as it takes no bits to communicate an event that always occurs. Now given a

source of m statistically independent random events { $E_1, \ldots E_m$ }, then the expected minimum number of bits to code the source is known as the entropy and computed as

$$H = -\sum_{i=0}^{m-1} P(E_i) log_2 P(E_i) .$$

One can interpret this to mean that without applying any type of lossless compression algorithm to the signal source, it is not possible to use less than H bits per E_i without losing information. If one compresses the source data with a lossless compression algorithm such as Huffman coding, one can expect an average lower bound of H bits per E_i[31, 36, 37].

2.1.2.10 Kullback-Leibler Divergence

Kullback-Leibler or relative entropy is derived from information theory and statistics, and is concerned with the statistical problem of discrimination by considering a measure of information between statistical populations, and is an important measure for pattern recognition and neural networks [38, 39]. The Kullback-Leibler divergence between two probability distributions P(x) and Q(x) defined over the same domain is

$$D_{KL}(P||Q) = \sum_x P(x) log \frac{P(x)}{Q(x)} \geq 0 .$$

2.1.2.11 Cross Entropy

Cross entropy [40] is a measure of the distance between two probability distributions. In machine learning, cross entropy is very commonly used as a cost function in training classifiers, and measures the average number of bits wasted by encoding events from an unknown distribution P with coding designed on a model of distribution Q. Given a source of m statistically independent random events { $E_0, \ldots E_{m-1}$ }, cross entropy is defined as

$$H = -\sum_{i=0}^{m-1} P(E_i) log_2\ Q(E_i)\ .$$

2.2 CLASSIFICATION OVERVIEW

Classification is generally the process of identifying discriminative features of an object to assign it to a category [31], and there is an immense need for such algorithms in cognitive RF architectures. Important characteristics of the classification features are that: they are invariant to translation; rotation; and the object scale. The choice of the features depends on the problem addressed, and the selected features must be insensitive to noise. The remainder of section this gives a general overview of classification techniques that can be used for waveform classification.

2.2.1 Matched Filters

The design of detectors from a known pattern is known as matched filtering, with the assumption that the optimal detector for a pattern must match the pattern. Matched filtering is applied to a wide range of problems, particularly with time-varying signals. The matched filter is a linear filter or template, h, which maximizes the signal to noise ratio (SNR) output of a linear detector $y[n]$ to an arbitrary input signal $x[t]$, and is defined by

$$y[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k]\ .$$

$y[n]$ is equivalent to convolving the unknown signal $x[t]$ with a conjugated time-reversed version of the template [31].

2.2.2 Decision Trees

Decision tree induction is one of the simplest and yet most useful forms of machine learning. A decision tree represents a function that takes as input a feature vector and returns a single value decision, where the input and output values can be discrete or continuous. There are many decision tree induction methods, with one of the earliest being Hunt's algorithm. The measures of tree node splitting can be based on several techniques such as the Gini Index, entropy, and misclassification errors[32, 41]. Research by Azzouz et. al. [42] has successfully used decision trees on statistical features for the classification of digitally modulated waveforms. A disadvantage of this technique is that the selection of statistical features requires expert knowledge of the waveforms.

2.2.3 Bayesian Decision Theory

Bayesian Decision Theory is a supervised learning technique based on the probability of occurrence and is used extensively for classification problems. Given an input feature vector x assumed to come from one of a set of probability distributions, Bayes Decision Theory determines which class the feature vector most probably belongs to. More specifically, given the known classes ω_i , such that

$$\omega_i \in \{\omega_1, \omega_1, \dots \omega_m\},$$

given feature vector x , such that

$$x = [x_1, x_2, \dots x_n]^T,$$

given a-priori class probabilities $p(\omega_i)$ such that

$$p(\omega_1), p(\omega_1), \dots p(\omega_m) ,$$

and given the likelihood of x with respect to class ω_i, such that

$$p(x|\omega_i), i = 1,2, \dots m ,$$

then, Bayes rule says that the probability of the class ω_i when given x is

$$p(\omega_i|x) = \frac{p(x|\omega_i) * p(\omega_i)}{p(x)}$$

where

$$p(x) = \sum_{i=1}^{m} p(x|\omega_i) * p(\omega_i) .$$

Then the classifier operates by assigning the pattern represented by the feature vector x to the most probable class such that

$$x \to \omega_i : \max_i p(\omega_i \mid x) .$$

Bayesian decision theory [31] makes no assumptions regarding the class population distributions, however in many applications the normal distribution is assumed. For example, to set up a Bayesian normal distribution model, the a-priori class probabilities $P(\omega_i)$ are computed by dividing the number of samples in each of the ω_i classes with the total number of samples. Then the likelihood estimates $p(x|\omega_i) = N(\mu_i, \Sigma_i)$, are calculated for each class ω_i by calculating the mean vector μ_i and correlation matrix Σ_i of the associated class samples.

2.2.4 Support Vector Machines

The support vector machine (SVM) is a supervised learning method for classification and regression, first identified by Vladimir Vapnik et. al. in 1992 [43], the SVM searches for a hyperplane in an N-dimensional space, where N corresponds to the number of features, that distinctly classifies the data points. Hyperplanes are decision boundaries between different classes of the data points. While there are many possible hyperplanes that could separate the classes, SVM finds a plane that maximizes the margin between classes. Support vectors are data points that lay on the hyperplane which maximizes the margin between classes. Support vectors define the position and the margin of the hyperplane.

Non-linear data that cannot be separated by a hyperplane in the number of dimensions as features, is transformed using kernel functions into a higher dimensional space to provide separation between the two classes in the higher dimensional space. Popular kernels used are: Polynomial, Sigmoid, Gaussian Radial Basis Function (RBF), and Laplace RBF. The SVM is considered a nonparametric technique because it relies on kernel functions and has successfully been used for signal classification using higher order statistics [44].

2.2.5 Normal Distribution Classifier

The well-known multivariate normal density function $p(x|\omega_i)$ is a reliable and well-known statistical approach for pattern classification problems. Given a feature space x and multiple normal probability distributions $p(x|\omega_i)$, where $i = 1 \dots N$ and N is the number of distinct

distributions, one can predict if input vector x is a member of the distribution ω_i. The general form of $p(x|\omega_i)$ for a normal distribution $N(\mu_i, \Sigma_i)$ is defined as

$$p(x|\omega_i) = \frac{1}{\sqrt{(2\pi)^k|\Sigma_i|}} \exp\left(-\frac{1}{2}(x-\mu_i)^T{\Sigma_i}^{-1}(x-\mu_i)\right),$$

where x is a k-dimensional column vector, Σ is the covariance matrix of the feature space, μ is the mean vector of the features, ${\Sigma_i}^{-1}$ is the inverse of the covariance matrix, and $|\Sigma_i|$ is the determinant of the covariance matrix [31].

The expression $(x-\mu_i)^T{\Sigma_i}^{-1}(x-\mu_i)$ is the well-known Mahalanobis distance providing a separation measurement between the vector x and $N(\mu_i, \Sigma_i)$, and will be referred to herein as $D_i(x)$. The population $N(\mu_i, \Sigma_i)$ with minimum $D_i(x)$ is the declared class:

$$\min_{i=1\ldots N} D_i(x), \quad x \in N(\mu_i, \Sigma_i).$$

The normal distribution classifier has many applications, for example it can be used for waveform classification and anomaly detection.

2.2.6 Deep Neural Networks

An artificial neural network (ANN) is [45] an interconnected group of artificial neurons inspired by the biological neural network. The connections between neurons have associated weights representing the strength of the connection. Each neuron computes its output as a function of the weighted sum of its input. The neurons in an ANN are organized in layers. A simple feedforward ANN has an input layer, an output layer that generates predictions and one or more

hidden layers that connect the input layer to the output layer. Neural networks are trained in a supervised manner by providing labeled data (inputs along with their associated output labels).

Training involves determining the error in prediction and using the error to determine which weights need to be adjusted and by how much. Prediction errors for a sample are calculated as the difference between the processed output of the network (often a prediction) and the target output label. Successive weight adjustment will cause the neural network to produce outputs that are increasingly like the target output. In training, one must decide upon sufficient training and test data set size to generalize the outcome of performance. Generalization is influenced by three primary factors: (1) the training sample size and how representative it is of the operating environment, (2) the neural network architecture, (3) and the problem complexity. In practice it seems that for good generalization performance, with a training size N, and fraction of desired error ε, the equation $N = W/\varepsilon$, where W is the number of free parameters (weights and biases). ε denotes the fraction of classification errors permitted during testing the data, for example with an error of 1%, the number of training examples should be 100 times the number of free parameters [46].

Deep neural networks (DNN) [47] are an extension of artificial neural networks (ANN) and generally considered as ANN structures with three or more layers. The biggest advantage of DNN architectures is that they provide for non-linear discriminators that can significantly improve performance over linear classifiers. This section gives an overview of some popular CNNs and how they can be used for waveform classification.

2.2.6.1 Convolutional Neural Network (CNN)

A CNN architecture [48] is an organization of several convolutional, activation, pooling, and other layers into one skeleton neural network architecture, where the values of each layer are calculated by function of its predecessor layer. CNN is called convolutional because the primary layers are the convolution layers that implement the standard mathematical convolution on the inputs at each layer.

Researchers in the wireless communications field have applied CNNs to cognitive radio classification tasks with some success. It has been shown that relatively simple CNN architectures have outperformed algorithms with decades of expert feature selections for radio modulation classifications [49].

Transfer learning techniques [50] also plays a significant role in allowing the use of previously developed CNNs from one domain and repurposing them for other domains. There are two key points associated with transfer learning. The first is that transfer learning makes use of pre-trained networks that were previously trained in different data domains. The second key point is the assumption that the original network was trained on a much larger data set than the size of the new target domain data set. A popular approach for transfer learning is fine tuning a linear classifier on top of a pre-trained CNN. This approach is carried out by removing the last layer of a pre-trained CNN and replacing it with a linear classifier layer where the number of output nodes equals the number of new target classes. All other original layers in the network are frozen in the sense that their weights and biases are not modified during the training process of the new target class. It is hypothesized that freezing the layers allows the new target training domain to take advantage of

the generalization that was learned in the previous larger training set. Studies have shown that a more refined approach that may yield better performance can be carried out by freezing only a subset of the layers, for example only the first three layers, replacing the last layer with linear classifiers, and training with the target data set. Due to the successes of transfer learning and layer freezing, much of the results discussed in this paper will utilize the technique of freezing all layers except the last for waveform classification and feature extraction.

The remainder of this section gives a brief overview of five successful CNN architectures that were developed in the ImageNet competitions that have provided high accuracies of performance. The CNN architectures AlexNet, GoogleNet, MobileNet, ResNet50, and Inception V3 each takes red-green-blue (RGB) input images and provides the output layer as 1,000 output nodes for image classification. These networks were selected as a sampling of the many CNNs developed for the ImageNet competitions. Table 1 shows the differences between the number of learnable parameters, the number of computational operations per forward pass of the CNN architectures with the learnable parameters, the number of layers, and the Top-1 accuracy results [51]. In general, CNNs with a greater number of operations tend to take longer to train. The learnable parameters are the number of weights and biases in the complete network. The Top-1 accuracy is the conventional accuracy calculation, or the total correct guesses out of the total number of samples per class.

Table 1. CNN Parameters and ImageNet Accuracy

CNN	Year	Learnable Parameters – million	Operations per single forward pass (G-ops)	Input Image Size	Number of layers	Top-1 accuracy [%] in ImageNet
AlexNet	2012	60	2.5G	227x227	8	54
GoogleNet	2014	5	3 G	224x224	22	68
Inception V3	2015	24	12 G	229x229	48	78
ResNet 50	2015	25	8 G	224x224	50	76
MobileNet	2018	3.4	0.3	224x224	54	75

The following subsections give a brief description of the architecture for each CNN.

2.2.6.1.1 AlexNet

In 2012, during the ImageNet competition AlexNet was the first entry that used a Deep Neural Network (DNN) to achieve the best performance at that time. The network is 8 layers deep, has about 60 million learnable parameters, and 2.5 Giga-operations. The network has an image input size of 227-by-227. Since the creation of AlexNet, studies have shown that despite its breakthrough performance, it is clearly oversized, and does not take full advantage of its potential learning ability. Since then, CNN architectures with much smaller networks and providing greater accuracy performance, such as GoogleNet, Inception, and ResNet have been created [51].

2.2.6.1.2 GoogleNet

In 2014, GoogleNet was designed to perform well under strict memory constraints and computational loads. GoogleNet is 22 layers deep and can classify images into 1000 object categories. The network has an image input size of 224-by-224. GoogleNet requires 3 Giga-

operations and has about 5 million learnable parameters which is a 12-times reduction with respect to AlexNet, which uses 60 million parameters [49].

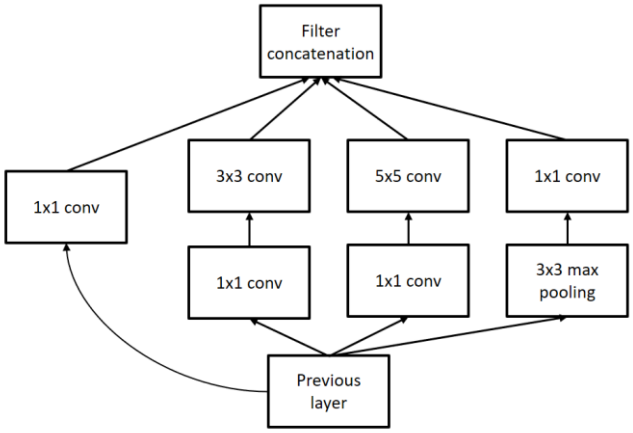


Figure 2. GoogleNet uses Inception units

GoogleNet uses the Inception units through the network as shown in Figure 2. ***GoogleNet uses Inception units*** to optimize performance. Each Inception unit contains four parallel paths with the output resulting from concatenation of the four parallel outputs. The first path is a bank of 1x1 convolutions that are a form of selective highway networks that simply pass information forward with no transformation. The second and third paths are 1x1 convolutions followed by 3x3 and 5x5 convolutions to provide multiple scales of feature detection. The last parallel path is a 3x3 pooling layer followed by 1x1 convolutions. Intermediate inception modules are connected to softmax layers that contribute to the network's global loss for training. The computational cost of Inception is much lower than its higher performing successors, which makes it more feasible to handle huge amounts of data needed for processing at reasonable cost or in systems where memory or computational capacity is limited [52, 53].

2.2.6.1.3 Inception V3

Inception v3, developed in 2015, is a CNN trained on more than a million images from the ImageNet database. The network is 48 layers deep, has an image input size of 299-by-299, about 24 million learnable parameters, and 12 Giga-operations. Inception V3 is a network architecture that has evolved from the continuation of research with the original Inception Units used in GoogleNet [52].

2.2.6.1.4 ResNet50

The residual network (ResNet) learning framework was designed to ease the training of deep neural networks. ResNet50 was released in 2015 by Microsoft Research Asia along with two other realizations, ResNet-101 and ResNet-152, and they obtained successful results in the ImageNet competition. The CNN is 50 layers deep, has about 25 million learnable parameters, and 8 Giga-operations, has an image input size of 224-by-224, and the layers are explicitly formulated using learning residual functions with reference to the layer inputs X. Studies have shown that the use of the $\mathcal{F}(X)$ residual network building blocks shown in Figure 3 are easier to optimize and can gain accuracy from increased depth.

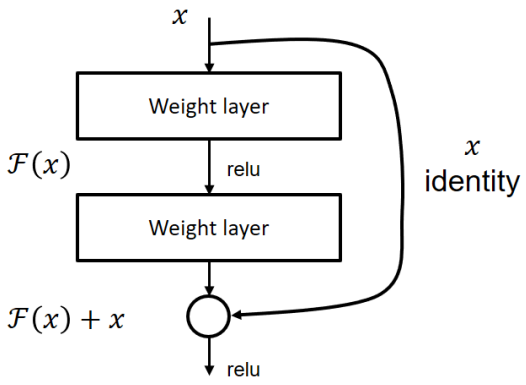


Figure 3. Residual network building blocks [54]

Denoting the desired underlying mapping as $H(X)$, the stacked nonlinear layers fit another mapping of $\mathcal{F}(x) \coloneqq H(x) - x$. The original mapping is converted into $\mathcal{F}(x) + x$. It is hypothesized that it is easier to optimize the residual mapping $\mathcal{F}(x)$ than to optimize the original x, unreferenced mapping, therefore allowing for the training of deeper neural networks with greater accuracy performance [54].

2.2.6.1.5 MobileNet Version 3

MobileNet is a CNN architecture developed by Google Inc. and is based on an inverted residual structure where shortcut connections are between the thin bottleneck layers. This neural network is especially tailored for mobile and resource constrained environments. The design significantly decreases the number of required operations and memory requirements while at the same time retaining the same accuracy as the more computation intensive CNNs. The design of MobileNet allows it to be rescaled in size for its targeted platform[55].

2.2.6.2 Generative Adversarial Networks (GAN)

GANs were originally designed as binary detectors but were then expanded to support multiclass classifiers. GAN is described as a network that has a generative model G that captures the data distribution statistics, and a discriminative model D that outputs a single scalar estimating the probability that a sample came from the training data rather than G. The generator is designed to generate new samples that are like the original samples. The discriminator is designed to inspect a sample and determine if it is real or fake. GAN is a generative process in the sense that it takes a machine that observes many samples from a distribution, and then it can create more samples from that same distribution. Mode collapse, a major problem in training highly complex GANs that has yet to be resolved, can occur when generators iterate through a small set of very plausible

outputs and the discriminator gets stuck in a local minimum as the generator over-optimizes for the discriminator[56], [57], [58].

There are many ways to compute cost function for the generator and discriminator and one of the simplest ways is to use the Minimax Game Cost Function that returns scalar values. Let $p_{data}(x)$ be a population of data where the statistical distribution is unknown, let $p_z(z)$ be a prior on input noise variables, let $G(z)$ be a generative model that takes samples from $p_z(z)$ and learns to estimate a model of the distribution of $p_{data}(x)$, and let $D(x)$ represent the probability that x came from the data distribution $p_{data}(x)$, rather than $p_z(z)$. With z and x as inputs, the cost function to be optimized is

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)} logD(x) + E_{z\sim p_z(z)} \log\left(1 - D(G(z))\right),$$

and is the cross entropy between the discriminator's predictions, and the correct labels in the binary classification task of discriminating real data from fake data.

2.2.6.3 Deep Associative Neural Networks

Deep associative neural networks (DANN) are designed for establishing associative memory by learning a probability model to capture data distributions using a hierarchical deep associative memory model based on unsupervised learning. The network model is optimized using a contrastive divergence algorithm with an iterative sampling method, and the network can achieve pattern recognition tasks with only one training session. After training such a network, when given new or corrupted data, the correct label or corrupted part is associated by the network [59].

2.2.6.4 Compressed Neural Networks

DNNs have achieved great success in many visual recognition and classification tasks. However, existing DNN models are computationally expensive and memory intensive, hindering deployment in devices with low memory resources. Therefore, there has been ongoing research to perform model compression and acceleration in DNNs without significantly decreasing performance. To address the complexity problems the approaches are divided into four categories: (1) parameter pruning and quantization, (2) low-rank factorization, (3) transferred/compact convolutional filters, (4) and knowledge distillation. These compression techniques have been successfully applied to popular CNNs such as AlexNet, VGG-16, and GoogleNet [60]. A successful design using compressed neural networks for joint antenna selection and hybrid beamforming allowed for overcoming complexity issues without significant degradation in performance [61].

2.2.6.5 Patching Deep Neural Networks for Nonstationary Environments

Patching Deep Neural Networks for Nonstationary Environments [62] discusses adaptive CNNs that have anomalous inputs. Instead of creating or updating a network to accommodate concept drift, the neural network patching concept leverages the inner layers of a previously trained network as well as its output to learn a patch that enhances the classification. Neural network patching is based on the concept that the original network can still classify a majority of instances well in non-stationary environments, and that the inner feature representations encoded in the network supports the classifier to cope with anomalous inputs. The author expresses that the approach is not addressing computational efficiency as part of the adaptive process. The patches

require on the fly training using the backpropagation training algorithm, which does not lend itself well to real-time adaptation applications[62].

2.2.7 Auto Encoder (AE)

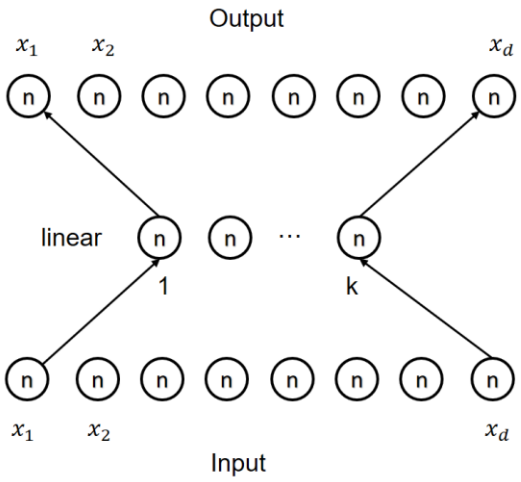


Figure 4. Three-layer neural network trained as auto-encoder [31]

A three-layer linear neural network trained as an auto-encoder is shown in Figure 4. ***Three-layer neural network trained as auto-encoder [31]***. The pattern of the training data set is presented to both input and the output layers, and the network is trained by gradient descent on a sum-squared error criterion. After the network is trained the top layer can be removed and the hidden layer provides the principal components for data compression or dimension reduction. The data can be uncompressed by using the top layer. Auto-encoding is a lossy compression algorithm. An extension of the AE is the variational autoencoder (VAE). It provides a probabilistic approach for describing an observation latent variable. AE and VAE are techniques that can be used for high dimensional feature reduction to reduce computational complexity in feature processing [31].

2.2.8 One Shot Learning

Learning classification models of object categories require large training examples. Research has shown that it is possible to learn much information about a category from just one, or a handful, of samples. The key insight is that, rather than learning from scratch, one can take advantage of knowledge coming from previously learned categories, no matter how different these categories might be. One technique uses a Bayesian implementation of one-shot learning. Object categories are represented by probabilistic models. Prior knowledge is represented as probability density functions on the parameters of the models. The posterior model for an object category is obtained by updating the prior in the light of one or more observations [63].

2.3 CLUSTERING OVERVIEW

Clustering is like classification but with the key difference that there are no predefined class labels, and is defined by Ahalya G. et al [64] as an analytical method to partition data into groups of identical objects where every group is defined as a cluster, which consists of objects that have an affinity within the cluster and a lack of similarity with the objects in other groups. The remainder of this section provides a brief overview of some well-known clustering methods.

2.3.1 K-Means

The conventional k-means clustering algorithm, with pseudo code shown in ***Table 2***, clusters a group of n-sample vectors x into K classes. The K-means algorithm starts with the population sample vectors, and makes a crucial assumption that there are K clusters with each having randomly chosen mean vectors μ_i. Each of the n-sample vectors x belongs to one of the K

clusters c, where c=1,2 ,…K [31]. The nearest μ_i cluster to each x_j sample is determined by computing the minimum distance between μ_i and x_j across all clusters and sample vectors. On each pass of the algorithm loop, each x_j sample is assigned to the closest μ_i cluster, and then each μ_i cluster position is recalculated by taking the mean of all x_j samples assigned to the μ_i cluster. The k-means algorithm iterates until there is no change in the μ_i mean vectors.

Table 2. K-means clustering algorithm

K-means Clustering algorithm

Begin

initialize n, k
randomly initialize $\mu_1, \mu_{12}, \ldots \mu_k$

do classify n samples according to nearest μ_i
$D(\boldsymbol{x}, \boldsymbol{\mu}) = \min_{j,i} \|\boldsymbol{x_j} - \mu_i\|$,
recompute μ_i
until no change in μ_i
return $\mu_1, \mu_{12}, \ldots \mu_k$

end

2.3.2 Dirichlet Process Mixture Model

The Dirichlet Process Mixture Model (DPMM) discussed by Yuelin Li et al [65] is a Bayesian nonparametric model (BNP) that addresses model complexity differently than conventional methods, for example, in k-means clustering, the model must fit data to K clusters, where K must be known a priori. However, DPMM is considered non-parametric because the number of parameters (or clusters) K to analyze the data are unknown. In theory, DPMM can represent an infinite number of parameters, and can be viewed as an extension to the Gaussian Mixture Model (GMM) because with GMM the number of parameters are assumed, but with DPMM more parameters can be "unfurled" as the data requires as it is presented to the model. The

DPMM model has been used in research for clustering analysis, however DPMM is a highly iterative process that requires all data points to be maintained in memory to arrive at an optimal clustering solution.

2.3.3 Self-Organizing Maps

Self-organizing-maps (SOM) [66, 67] [68, 69] are among the most well-known, unsupervised competitive learning network approaches to produce low-dimensional, discretized representations of the input training samples and for efficiently clustering high dimensional datasets, with a primary advantage in that the number of clusters are not assumed. SOM are network structures proposed by Kohonen that combines competitive learning principles with a topological structuring of nodes such that adjacent nodes tend to have similar weight vectors. SOM architectures has been applied to many domains such as speech recognition, text clustering, robotics, satellite, and medical image classification. The pseudocode for the SOM algorithm is shown in Table 3.

Table 3. Pseudocode for SOM

Pseudo Code: Algorithm Self-Organize:

Select network topology
Initialize weights to small random variables
Initialize current neighborhood distance D(0) to positive integer

While compute bounds not exceeded **do**

1. Select an input i_l
2. Compute the square of the Euclidian distance i_l from weight vector (w_l) associated with each output node
3. Select the output node j with minimum error
4. Update weights to all nodes within a topological distance

End-While

The interest in this research in exploring the use of SOM structures stems from their ability to classify data, and in their online ability to self-learn new classes in non-stationary environments,

making such techniques attractive approaches for classifying RF waveforms in non-stationary environments. The remainder of this section discusses some variants of SOM architectures of interest to this research.

2.3.3.1 Neural Gas (NG)

NG is a well-known clustering algorithm that assumes several centers in R^n and successively inserts topological connections among them by evaluating input signals drawn from a data distribution. The fundamental principle is for each input x, connect the two closest centroids by an edge. Centers lying on the input dataset or in its vicinity develop edges. NG operates on an adaptive process whereby each input signal adapts the k nearest centers where k is decreasing from a large initial value to a final small value [70, 71].

2.3.3.2 Growing Neural Gas (GNG)

GNG defines an unsupervised learning incremental network model that can learn the topological structure in each set of input vectors by means of a Hebbian learning rule. This model can continuously learn, adding nodes and connections until a performance criterion is met. One important application of GNG is dimensionality reduction or in finding a low-dimensional subspace of the input vector. The applications of GNG are broad and include vector quantization, clustering, and interpolation. GNG represents an extension of the NG algorithm in which the number of neurons is not fixed but may grow over time. This is particularly useful in clustering tasks where the number of clusters is previously unknown [70, 72].

2.3.3.3 Online Semi-supervised Growing Neural Gas (OSSGNG)

OSSGNG is an online semi-supervised classification approach based on Growing Neural Gas (GNG). The semi-supervised classification approach based on GNG requires that the training data be explicitly stored as the labeling is performed after the training phase. The OSSGNG approach relies on online labeling and prediction functions to process labeled and unlabeled data and in an online fashion, without the need to store any of the training data. This method has shown that using on-the-fly labeling strategies does not significantly degrade the performance of classifiers based on GNG, while eliminating the storage of training examples. It has been shown that OSSGNG performs as good as the previous semi-supervised extensions of GNG which rely on offline labeling strategies [72].

2.3.3.4 Online Growing Neural Gas (OGNG)

OGNG developed by Beyer [1], extends GNG with an on-the-fly labeling and prediction step, therefore making GNG capable of learning from continuous data streams where the labeling process is performed online and only neurons are stored, but no explicit training examples are stored. GNG is extended to OGNG by a step in which the label of the presented stimulus is assigned to the winner neuron during the learning process.

2.3.3.5 Dynamic Online Growing Neural Gas (DYNG)

Per the works of Beyer, DYNG is a streaming data classification approach is an extension of OGNG. DYNG exploits labelled data during processing to adapt the network structure as well as the speed of growth of the network to the requirements of the classification task. It speeds up learning for new categories and reduces growth of the subnetwork representing the category once

the classification performance converges. DYNG addresses concepts of incremental learning and online learning. Incremental learning deals with incorporating existing knowledge into an existing model, and online learning addresses stepwise model updates and does not need to explicitly store the training examples. These two challenges are addressed by DYNG and are critical for systems to learn and adapt on the fly. DYNG is designed to work with non-stationary data and category distributions and can cluster new categories on the fly when they are supplied with the appropriate label. DYNG is an approach that utilizes label information to quickly adapt to non-stationary data in a classification task. Finally, DYNG is designed with three key challenges of non-stationary data in mind. 1) Concept drift. 2) Efficiency 3) Plasticity and Stability. Table 4 shows pseudo code for the DYNG algorithm [1, 73].

Table 4. Beyer, DYNG [1, 73]

1. Start with two units n_i and n_j at random positions in the input space.
2. While Stream.hasNext do
3. $x_i := Stream.next(t)$ with $x_i \in R^n$
4. Find the nearest unit n_1 and the second nearest unit n_2

//Steps 5-8 describe the fast mapping strategy. In these steps, a new neuron is inserted into the framework at position of the //stimulus. The new neuron adopts the label of the stimulus.

5. if $l^t(x_i)$ is unknown then
6. Add a neuron n_{new} with $n_{new} := x_i$ and $l^t(x_{new}) := l^t(x_i)$
7. Create an edge between $n_{new}$and n_1
8. end if

9. Update local error variable of n_1 and increment the age of all edges emanating from n_1 according to OGNG
10. if $l^t(n_1) \neq l^t(x_i)$ then
11. Update the class error: $\Delta ClassError^t(c_1) = 1 - \frac{\Delta error^t(n_1)}{max_{n \in N}\,(error^t(n))}$, with $l^t(n_1) = c_1$
12. end if

//step thirteen relabeling method appears to cause a "logical" problem with the if statement //on steps 15-23. Namely that since step 13 assigns the label of the n1 neuron equal to the //label of the stimulus, that means that the "if" statement on line 15 never holds, therefore //the distance based insertion never occurs. See line 15.

13. $l^t(n_1) := l^t(x_i)$ {OGNG relabel-method}
14. Move n_1 and all its topological neighbors according to OGNG and connect n_1 and n_2

//steps 15-23, Distance based insertion strategy.

15. if $l^t(n_1) \neq l^t(x_i) \wedge imp\big(l^t(x_i)\big) = "true"$
16. Find nearest units n_{lb} with $l^t(n_{lb}) = l^t(x_i)$
17. if $\left|\omega_{n_{lb}} - x_i\right|^2 \geq \left|\omega_{n_{lb}} - x_i\right|^2 \tau + \left|\omega_{n_1} - x_i\right|^2$ then
18. Add a neuron n_{new} with $\omega_{n_{new}} = x_i$ and $l^t(n_{new}) = l^t(x_i)$
19. Create and edge between n_{new} and n_{lb}
20. else
21. Move n_{lb} towards x_i by the fraction of e_b: $\Delta\omega_{n_{lb}} = e_b(x_i - \omega_{n_{lb}})$
22. end if

```
    end if
    Remove edges and neurons according to OGNG

//steps 25-34, the classification error of all known categories at iteration t is compared to those,
//at iteration −λ . If there is an improvement (by decrease in error), then the improvement
//flag for the particular class is set to "true" to indicate improvement of classification performance
//of the particular class, otherwise it is set to false. Additionally, insert new neuron according to
//OGNG.
    if t mod λ = 0 then
      for all classes c_i ϵ C do
        if classError^{t−λ}(c_i) > classError^t(c_i) then
          imp(c_i) = "true"
        else
          imp(c_i) = "false"
        end if
      end for
      Insert a new neuron and update the network link structure according to OGNG
    end if

    Decrease all local error variables of all neurons n_i and all class error variables of all classes c_i by a factor β
    t++
end while
```

The DYNG algorithm is of great interest to this research because of its characteristics of quick growth of the network and its benefits on life-long learning in non-stationary environments. However, a key shortcoming of DYNG is that it operates in either a training or prediction mode and does provide an inherent process for switching between these two modes. This shortcoming has been resolved in this research by extending DYNG with an anomaly detection and one-shot learning algorithms to recognize when an input stimulus is an outlier of the known classes represented by the DYNG network, and to automatically switch DYNG between the learning and prediction modes while online.

2.4 ANOMALY DETECTION

Anomaly detection [74] [75] is the ability to identify unexpected events in datasets which differ from normal expectations and is often applied to detect unlabeled data, and it is a crucial component in notifying cognitive classifier systems when unlabeled data enters the system so that appropriate actions can be taken in response to the anomaly. There are three learning modes for

training anomaly detectors: 1) supervised anomaly detection is where the data is fully labeled and consists of training and test data where a classifier is trained first and then applied later. 2) Semi-supervised anomaly detection uses training and test datasets, where the training data only consists of the normal data without any anomalies. 3) Unsupervised anomaly detection is the most flexible case that does not require any labels, and there is no distinction between a training and a test dataset. The data streams input to anomaly detectors can be either discrete or continuous in nature, for example discrete data may be used to monitor network traffic data packets for intrusion detection, whereas continuous data may be used to recognize the presence of unlabeled waveforms in a continuously time-sampled wireless system.

The remainder of the section gives a brief survey of anomaly detection techniques for continuous and discrete data streams that may be of use to this research.

2.4.1 Normal Distributions and Principal Component Analysis

Anomaly detectors [21] are often implemented by modeling multivariable populations with multivariate normal distributions. Once a model of the population is established, one can implement an anomaly detector using distance metrics to determine when a sample's distance falls outside established boundaries. A computational complexity problem can arise when the number of feature variables are of extremely high dimension, therefore Principal Component Analysis (PCA) is considered a popular approach to reduce the computational complexity by performing anomaly detection in a lower dimensional space [76]. PCA anomaly detection became of interest in this research because the waveform feature vectors output by the CNN architectures are very large, for example of length 4096, and for a multivariate normal distribution model with Mahalanobis

distance measure would require a computationally expensive 4096-by-4096 covariance matrix which may not lend itself to processing under real-time constraints of embedded systems.

2.4.2 Covariance Analysis and Sliding Window

A sliding window technique [77, 78] using a covariance matrix is applied to detect anomalies due to distributed denial of service (DDoS) attacks on streaming network traffic. The amount of data used to establish a covariance matrix is often a bottleneck in time, so the use of a sliding window covariance matrix is used to cope with the large number of network data inputs. The technique first creates an expected covariance matrix from a set of training data vectors with p features. Then when online, n random sample vectors are acquired and used to compute a second covariance matrix, where n is the length of the sliding window with the size being selected to keep up with the data rates. A distance formula is then applied between the two matrices and an anomaly is declared if the distance exceeds a set threshold. Periodically and application dependent, the expected covariance matrix is updated to adjust expectations to the dynamics of the non-stationary environment.

2.4.3 Concept Drift Detectors

Concept drift [79] is defined as streaming data distributions that are non-stationary and evolve over time, including cases where the distribution of labeled classes becomes non-stationary. To perform classification in such environments, it is a significant challenge to use incoming samples to keep the learning classification model updated to recognize the drift and to correctly modify the recognition system to maintain performance. Research in the field targets complicated problems such as how to accurately detect concept drift in unstructured and noisy datasets.

A vital component of concept drift systems [27] are the drift detectors used by active learners to accumulate enough new and useful samples to update the classifier during drift detection. Drift detectors, also viewed as anomaly detectors, are external modules that identify a moment when change appears or is likely to appear.

The drift detectors fall into three categories: 1) error-rate based 2) data-distribution based; 3) and multiple-hypothesis testing. The error rate-based drift detectors form the largest category of algorithms and focus on tracking changes in the online error rate of base classifiers. The second largest category of drift detection algorithms is data distribution-based, where the algorithms use distance metrics to quantify dissimilarity between the distribution of historical data and the new data, and these algorithms address concept drift from the root sources by identifying the time and location of the drift within the distribution. The data distribution-based algorithms tend to incur a higher computational cost than error rate-based drift detectors. Finally, the multiple hypothesis test drift detectors apply similar techniques to the previous two categories and uses multiple hypothesis testing to detect concept drift.

2.4.4 OGNG for Anomaly Detection

OGNG has been demonstrated as a network for performing online anomaly detection in changing surveillance scenes [80], and is performed with the assumption that OGNG summarizes the high-density region of behavior space into clusters of neurons, and it is not expected for an anomalous stimulus to be close to any one of these clusters. The internal structure of OGNG is used for the anomaly detection by looking for patterns far away from the neurons and labeling them as anomalous, for which far away is measured by a time-varying threshold. This is done by searching

for the closest DYNG neuron to the input stimulus, and then declaring an anomaly if the distance between them exceeds an overflow threshold, and to reduce false alarms in noisy environments, several declared anomalous inputs are averaged together for decision making. This anomaly detection technique relies on making use of the mean feature vectors and standard deviation statistics associated with each of the OGNG neurons.

2.4.5 One-Class Neural Network Anomaly Detector

Work by R. Chalpathy et.al. [81] developed a one-class neural network (OC-NN) architecture to detect anomalies in complex data using the concept of creating a tight envelope around the normal data, by designing the neural networks hidden layer for anomaly detection. This is a departure from suboptimal hybrid approaches that are unable to influence representational learning in the hidden layers, for example by using a feature learning autoencoder and then inputting the features into a separate anomaly detection method such as one-class SVM (OC-SVM). The OC-NN has been shown to outperform OV-SVM and other hybrid methods, by combining the ability of deep networks to extract progressively rich representations of data along with the one-class objective to implement a hyperplane to separate all the normal data points from the origin.

2.5 OVERVIEW OF COGNITIVE RADIO AND RADAR SYSTEMS

The electromagnetic spectrum provides the primary means of operation for radar and radio systems. Radio waveforms are modulated with relatively low bandwidth information bearing signals. The modulated signals are then up mixed to much higher carrier frequencies for transmission. A key reason for mixing modulated waveforms to a desired carrier frequency is because the terrestrial propagation range of RF waveforms is heavily influenced by the carrier

frequency. To take advantage of frequency verses transmission range, communications systems selectively decide which electromagnetic bands to operate in by mixing the modulated waveform up to the desired carrier frequency band.

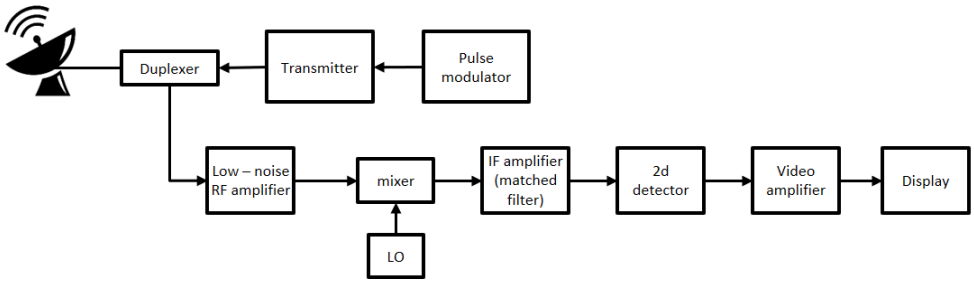


Figure 5. Block diagram of a traditional pulse radar[82]

The traditional pulse radar shown in Figure 5 has a duplexer with tight timing and control signals allowing for the transmission and reception of waveforms on the same antenna structure. For transmission, a pulse modulated waveform is input to the transmitter, and on reception, a low noise amplifier first amplifies the received carrier signal. The signal is then fed to a local oscillator (LO) mixer to bring the signal down to a lower frequency band. The mixer output is then fed to an intermediate matched filter to extract only the frequencies of interest. The signal is then extracted at the second detector (2d) and amplified by the video amplifier where it can be displayed.

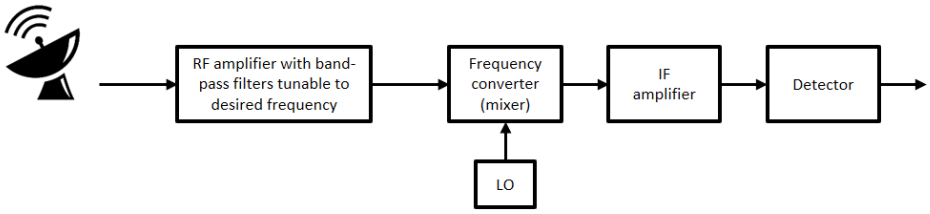


Figure 6. Superheterodyne communications receiver[83]

The traditional superheterodyne communications receiver is illustrated in Figure 6. The receiver consists of an RF section, a frequency converter, an intermediate frequency (IF) amplifier, and a detector. The RF section is a tunable filter and an amplifier that receives the desired frequency by tuning the filter to the desired frequency band. The LO frequency translates the high frequency carrier signal to a fixed frequency range acceptable to the IF amplifier. The IF amplifier amplifies the signal to an acceptable level for the detector, and finally the detector's output will be used to extract the signal information [83]. A superheterodyne transmitter does the reverse process of the receiver to transmit the desired signals.

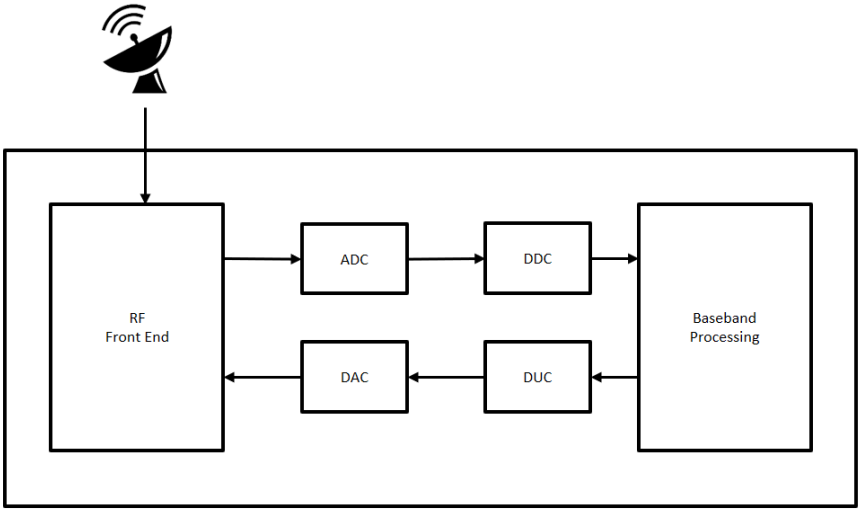


Figure 7. Software Defined Digital Radio[84]

A common limitation of traditional transceivers is they are not adaptable in the sense that once designed for specific functions, their general functionality remains static, therefore limiting their flexibility in being repurposed. The Software Defined Radio (SDR) depicted in Figure 7 are more adaptable to various formats and protocols. SDR operates in multiband and multimode

configurations with the ability to adapt as defined through software. A transceiver configuration is an SDR if its communications functions are implemented with programs running on a hardware processor. The same platform can be used to support a multitude of algorithm standards and protocols. The SDR consists of three main components - an RF Front End section, an IF section, and a baseband section. The analog to digital converter (ADC) performs on receive, and the digital to analog converter (DAC) performs on transmit. The digital down conversion (DDC) supports demodulation on the receiver, and the digital up-converter (DUC) supports modulation on transmission. Much of the digital processing techniques performed within SDRs were traditionally performed in discrete analog devices in traditional transceiver architectures[84]. Therefore, as devices have become faster and smaller much of the processing functionality is being pushed into application specific integrated circuit (ASIC) devices, highly configurable field programmable gate array (FPGA) devices, or software controlled digital signal processing (DSP) devices.

The flexibility of SDR devices makes them extremely attractive for exploring cognitive RF systems. Cognitive radios [20] are intelligent communication systems that can autonomously modify their transmission parameters such as operation frequency, modulation, transmit power, and communication protocols based on the interactions with their environment.

The National Institute of Health defines cognition as that "conscious mental activity that informs a person about his or her environment. Cognitive actions include perceiving, thinking, reasoning, judging, problem solving, and remembering." It is surmised that cognitive ability or knowledge aided processing can provide invaluable information in the decision-making process of cognitive radars [85], and the original term of "cognitive radar" was introduced for the first time by Haykin in 2006 [17].

The idea of cognitive radio was originally perceived by Mitola in 1999 [17, 86] as a programmable and adaptable communications device that can passively learn operator preferences and dynamically adapt to the operating environment. The FCC eventually identified cognitive radio as a way for users to access underutilized spectrum reserved for licensed or primary users. As a result, cognitive radio became an important research area viewed by the RF community to improve the underutilization of the electromagnetic spectrum for reliable communications. In 2005 Haykin, [15] defined in great detail his definition of a cognitive radio.

Cognitive RF systems can be viewed as having three major components: 1) the ability to perceive; 2) the ability to think and reason; 3) and the ability to remember. These components work together through a feedback loop to provide a perception to action operating cycle. For RF cognition, perception is carried out as spectrum sensing, to think and reason is implemented with machine learning, expert and reasoning systems, and the ability to remember is implemented with memory or database components. Cognition may entail cognitive control of the antenna system, the receiver/exciter hardware, waveform generation, and the signal processing algorithms. Combining these components can lead to systems capable of learning from their operating environment and performing autonomous reasoning to provide enhanced performance over conventional systems. At a higher level of integration, cognitive RF systems can be networked together, providing a collective level of reasoning that may yield greater performance than possible with a single cognitive system. From a hierarchical perspective, cognitive RF systems can be viewed as having five layers of abstraction from highest to lowest - the application layer, the architecture layer, the modes of operation layer, the cognitive processing layer, and the low-level underlying algorithms [87]. As an example, for traditional radar configuration, parameters typically involve the carrier frequency, pulse repetition frequency, duty cycle, bandwidth, waveform, coherent processing interval, dwell time and beam angles. Once configured, these settings are

typically kept static during radar operation. However, with a cognitive radar system, the radar can autonomously adjust its parameters to optimize performance.

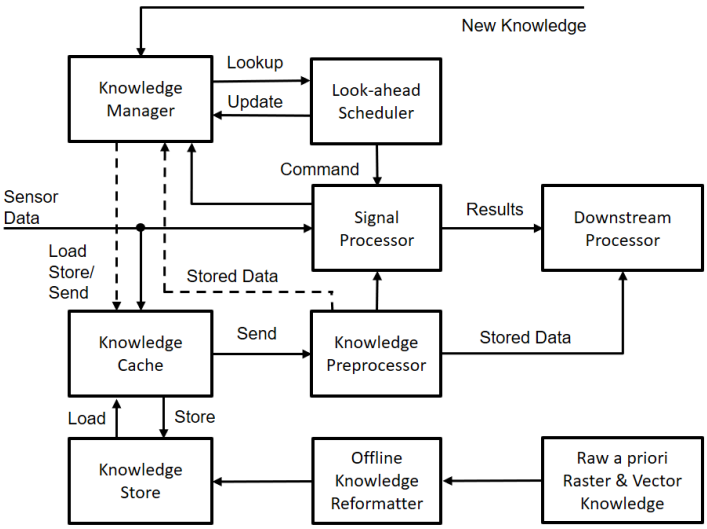


Figure 8. KASSPER Architecture [88]

A cognitive system called the Knowledge Aided Sensor Signal Processing and Expert Reasoning (KASSPER) Real-Time Signal Processing Architecture defined by Lincoln Laboratory (in 2004) [88] is shown in the block diagram in Figure 8. The key components defined by KASSPER are New Knowledge, Signal Processor, Knowledge Preprocessor, Knowledge Cache, and Dynamic Knowledge, and these components are focused on real-time, low complexity, and low latency aspects. The other components can be viewed as higher level system management functions without real-time constraints.

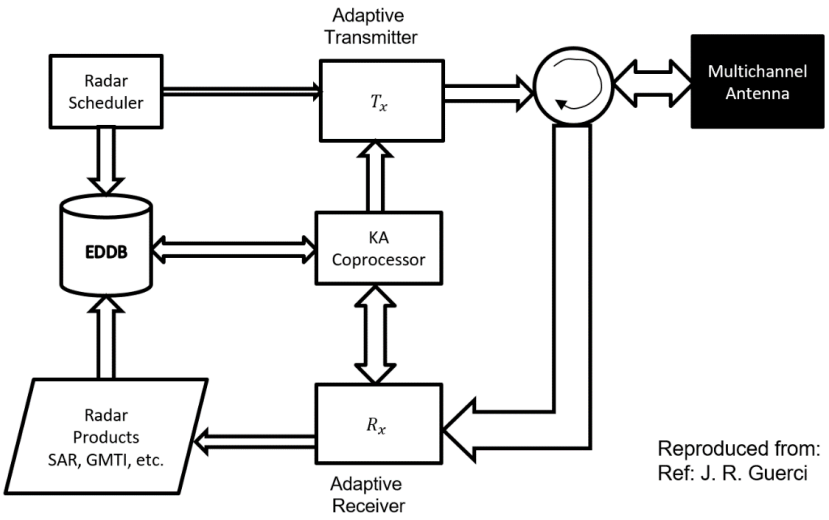


Figure 9. KASSPER in a Real-Time Ground Moving Target Indicator (GMTI) Radar System [85]

In 2001 DARPA and the Air Force Research Laboratory (AFRL) utilized KASSPER in a real-time airborne GMTI radar system [85]. This research has shown performance improvements obtained using this architecture following an airborne racetrack flight path of the area of interest. Figure 9 shows a block diagram of the GMTI radar using the KASSPER architecture consisting of radar scheduler, environmental dynamic database (EDDB), adaptive transmitter, adaptive receiver, knowledge aided (KA) coprocessor, and multichannel antenna (phased-array antenna). The EDDB contained a priori knowledge such as terrain maps and geographical objects such as building structures, as well as dynamically generated information from the radar sensors.

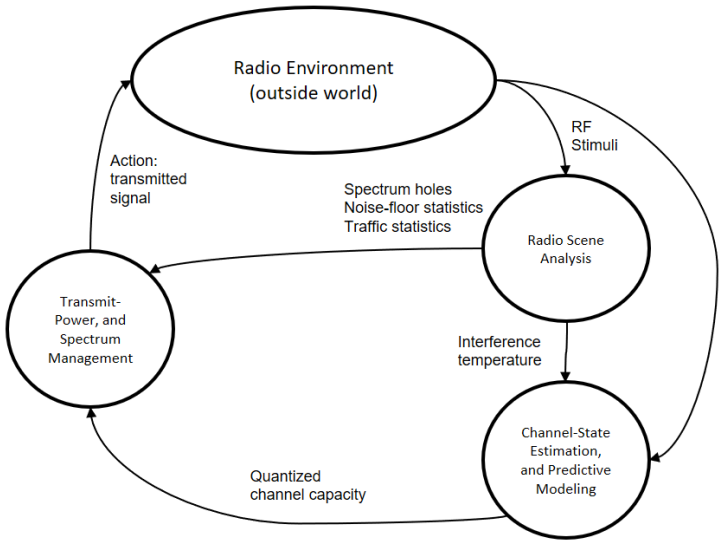


Figure 10. Haykin Cognitive Radio – The Basic Cognitive Cycle [15]

In 2005, Haykin authored a paper titled "Cognitive Radio: Brain-Empowered Wireless Communications" [15], with the position that cognitive radio is a novel approach for improving the utilization of the electromagnetic spectrum, and furthermore is assumed to be implemented using software defined radio technology. Two primary objectives to support cognitive radios were defined, the first was to provide reliable communications whenever and wherever required, and the second objective was to make efficient use of the electromagnetic spectrum, and this architecture defined a basic cognitive cycle as shown in Figure 13. ***Three-layer model of cognitive radar architecture [91]***. The cognitive cycle consists of a radio-scene analyzer to estimate the interference temperature of the radio environment and to detect spectrum holes. The cognitive cycle also consists of a channel-state estimation model for the prediction of channel capacity for use by the transmitter. This model can be used for higher level reasoning such as adaptive beam forming to control the beam width and pointing angles of phased array antennas to prevent interference.

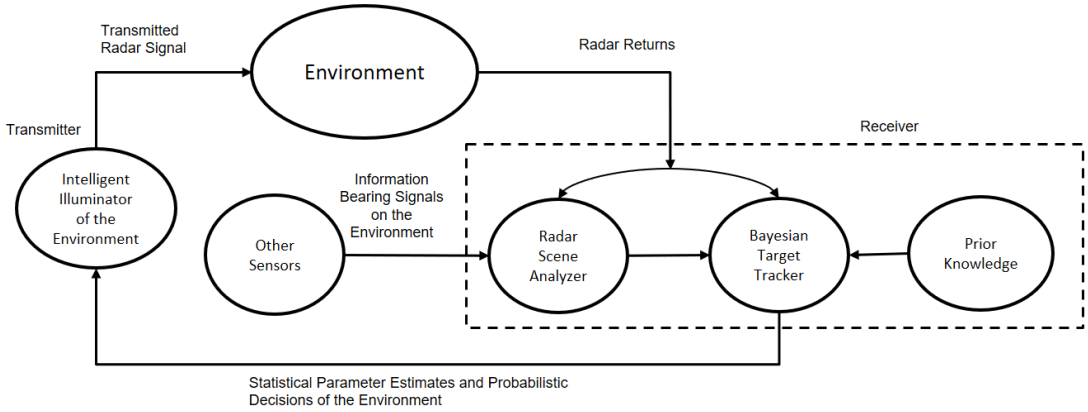


Figure 11. Haykin Cognitive Radar, viewed as a dynamic closed-loop feedback system [89]

In 2006 Haykin introduced a new concept of a dynamic closed loop feedback system called a "cognitive radar" [89], with block diagram shown in Figure 11. Three main components were identified to implement such a system: 1) intelligent signal processing; 2) feedback from the receiver to the transmitter; and 3) preservation of the information content of radar returns which is realized by the Bayesian approach for target detection and tracking. Haykin further defined cognitive radar with three additional distinct capabilities: the ability of the radar to sense its environment; the ability of phased-array antennas to electronically scan the environment in a fast manner; and the use of high-speed computers to digitally process signals in real-time. Haykin's cognitive radar approach uses the idea of Bayesian state estimation with probabilistic modeling of alternative states, carried out using cognitive signal processing cycles.

The cognitive cycle shown in Figure 11 is composed of a closed loop radar processing cycle with the following components; the radar transmitter composed of an intelligent environment illuminator; radar returns received by the non-stationary environment dispersed into two paths, the radar scene analyzer, and the Bayesian target tracker; the output of the Bayesian tracker then feeds back to the intelligent illuminator to make adjustment to the transmitter. Furthermore, Haykin views cognition from a two-way perspective. The first is viewed as inside out and is represented

by prior knowledge of the environment. The other is viewed as outside-in and may be viewed as short-term memory, produced on the fly by the receiver from the real-time information bearing signals. Haykin further continues with a detailed discussion of the Bayesian Target Tracker providing a concrete example of how such a tracker can be implemented in hardware.

In 2015 Kristine Bell et al [90], provided a rigorous mathematical formulation of a "Cognitive Framework for Target Detection and Tracking". The model includes a higher-level tracking processor and defines feedback mechanisms and optimization criterion to obtain the next set of sensor data. The framework mimics the perception-action cycle and includes sensing in the transmitter and receiver. The formulation includes processing in the detector and tracker, perception in the conversion and sensor data, memory for past data behavior, prediction capability, and a controller for decision making.

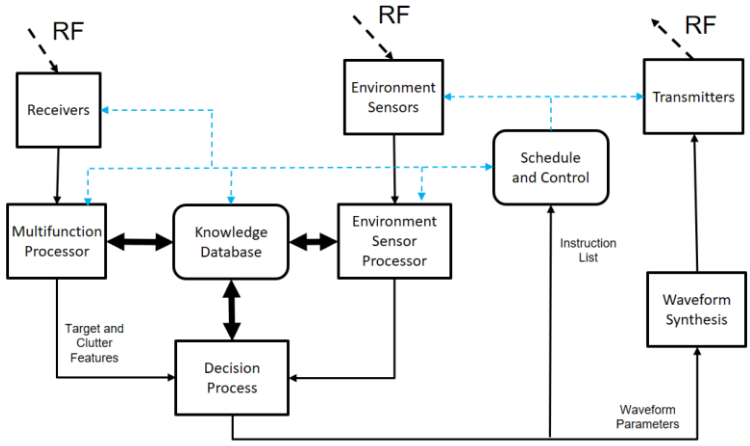


Figure 12. Generalized Cognitive Radar Framework - Martone [17]

Figure 12 shows the Generalized Cognitive Radar Framework envisioned by Martone. Martone identifies six principal components of a framework 1) informed decision making; 2) passive environmental sensors and radar sensors; 3) machine learning algorithms to improve performance and adapt to unknown environmental scenarios; 4) a knowledge database containing environmental clutter target, and a priori information; 5) a waveform synthesizer for targets of interest; and 6) receiver to transmitter feedback to mitigate clutter/interference and maximize target information. [17].

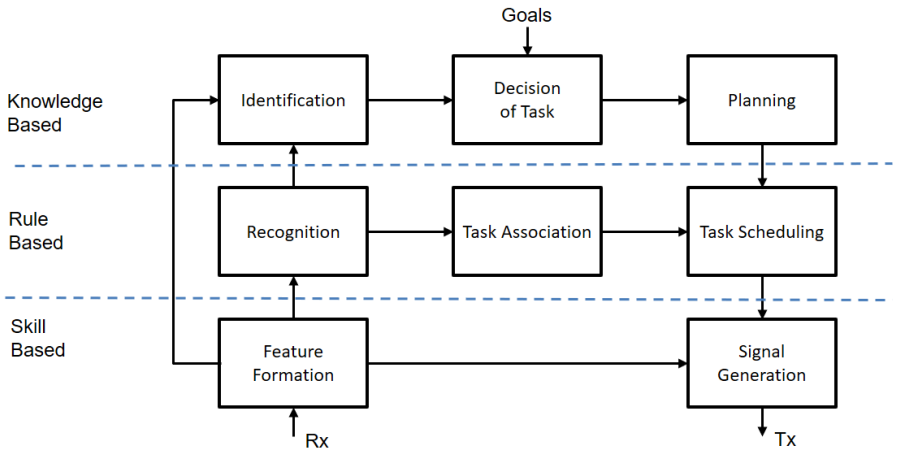


Figure 13. Three-layer model of cognitive radar architecture [91]

Figure 13. ***Three-layer model of cognitive radar architecture [91]*** depicts a cognitive radar architecture defined by Bruggenwirth [91], viewed as a three-layer model of human cognition, with the layers ordered from top to bottom by decreasing levels of reaction time and increasing levels of abstraction. The skill-based layer speaks to the subconscious, sensor-motoric aptitudes. The rule-based layer speaks to responses that a human can do intentionally in known circumstances. The highest level of cognitive behavior is called knowledge-based and provides objective and goal-oriented actions in new circumstances. In this framework, unique cognitive sub-functions are

mapped onto signal processing and artificial intelligence algorithms to increase the automation and information abstraction level of a radar.

2.5.1 Radio Frequency Waveforms

This section gives a brief overview of the fundamental building blocks of modulation and demodulation schemes used for generating digital waveforms for radio and radar systems. The material in this section is derived from [83, 92].

2.5.1.1 Components of a Sine Wave

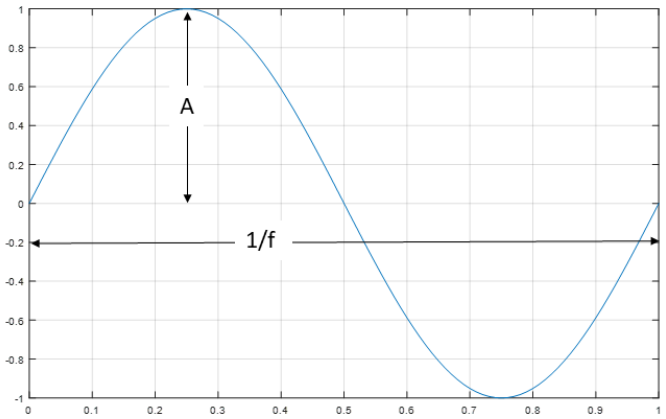


Figure 14. Components of a Sinewave

Figure 14 shows the basic components of a sinewave $V(t) = A * sin(2\pi ft + \theta)$, where A is the peak or amplitude, f is the frequency, θ is the phase shift, and t is the time variable. To perform any form of waveform modulation requires altering one, two, or all three of the parameters A, f, and θ.

2.5.1.2 Simple Amplitude Modulation (AM)

An example of AM is where the amplitude of a high frequency carrier sine wave is modulated by a much lower frequency information signal. The AM modulation scheme is defined as $V(t) = A(t) * \big(sin(2\pi ft + \theta)\big)$, with the higher carrier frequency component defined $sin(2\pi ft + \theta)$. The information bearing signal $A(t)$ is of much lower frequency bandwidth than the carrier, and the effect of multiplying the information signal times the carrier is that the carrier amplitude will vary as a function of the message amplitude. For example, in a case where $A(t) = sin(2\pi ft + \theta)$) as shown in Figure 15, and the carrier is defined as $C(t) = sin(2\pi * 10 * ft + \theta)$ shown in Figure 16, the carrier $C(t)$ frequency is 10 times the message A(t) frequency.

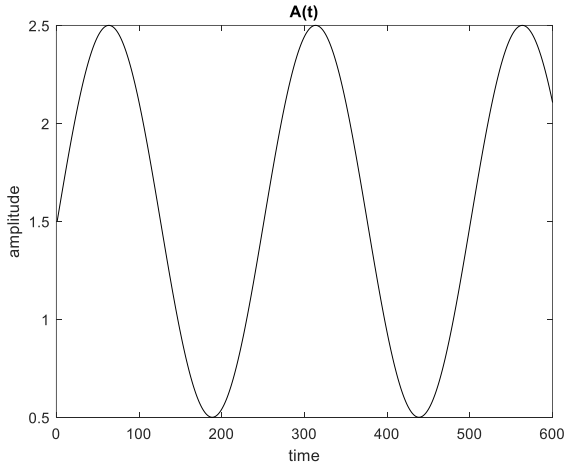


Figure 15. 1 $A(t) = sin(2\pi ft + \theta)$)
low frequency message sine wave

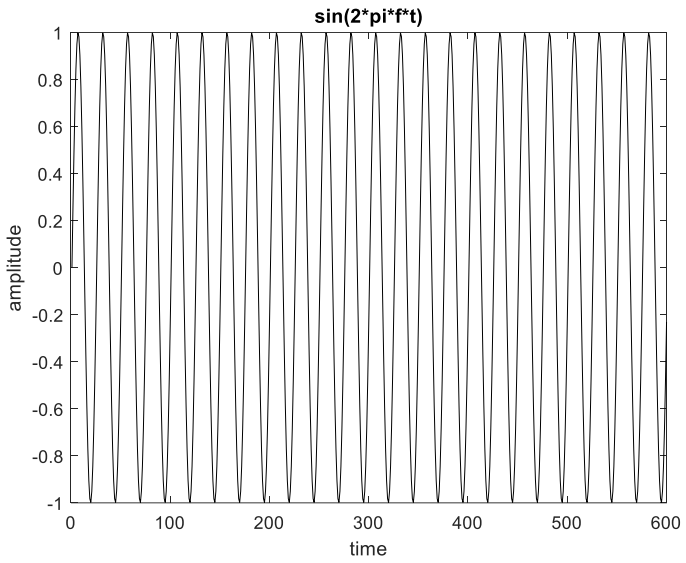


Figure 16. $C(t) = sin(2\pi * 10 * ft + \theta)$)
high frequency carrier sine wave

Shown in Figure 17 is the result of multiplying the message and the carrier signals. The amplitude of the result is varying as a function of the low frequency message. The amplitude variation is referred to as the envelope of the signal and indicated by the dotted lines. This envelope takes on the shape of the low frequency message. Upon reception, one can recover the original message by using a low pass filter.

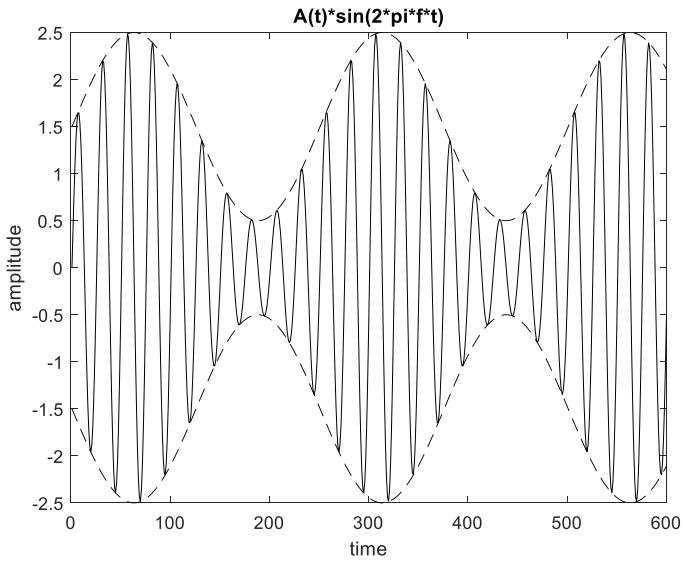


Figure 17. AM waveform

AM is a simple case of waveform modulation still in use today for radio communications. The amplitude of the carrier is modified to carry the information on the amplitude of the transmitted signal. In practice it is simply not possible to just transmit the original message A(t) through space with a carrier. The use of a higher frequency carrier is necessary due to the physics of how electromagnetic signals travel through physical mediums such as antennas, and how electromagnetic waves propagate through space as a function of carrier frequencies.

2.5.1.3 Quadrature Signals

The concept of quadrature signals is fundamental to digital waveform modulation and entails the phase (or angular) relationships between two signals. Two signals are defined as quadrature when they are 90 degrees apart in phase. For example, the sine wave $I * cos(2\pi ft)$and cosine wave $Q * sin(2\pi ft)$ are in quadrature with one another meaning they are 90 degrees apart in phase. The amplitude of the cosine is referred to as the I or in-phase signal, and the amplitude of the sine waveform or quadrature signal is noted as Q.

Figure 18 shows the relationship between I and Q with the sine and cosine having the same amplitude. In practice it is the relationship between the phase shift and amplitude differences between I and Q that is utilized to convey information within RF waveforms.

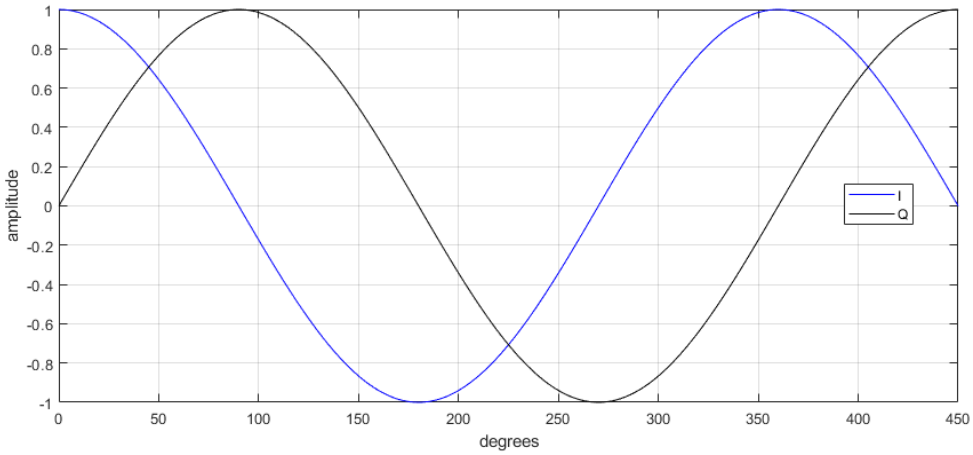


Figure 18. Plot showing quadrature (Q) and in phase (I) relationship

2.5.1.4 Properties of summing quadrature signals

To further extend these ideas, instead of I and Q being constant, the amplitude of the in-phase and quadrature components can be varied as a function of time, yielding the in-phase component as $I(t) * \cos(2\pi ft)$ and the quadrature component as $Q(t) * \sin(2\pi ft)$, therefore allowing for possibilities of more complex modulation schemes. Figure 19 shows the result of adding the time varying I and Q components together, resulting in a larger amplitude sinewave with a 45-degree phase shift shown in red.

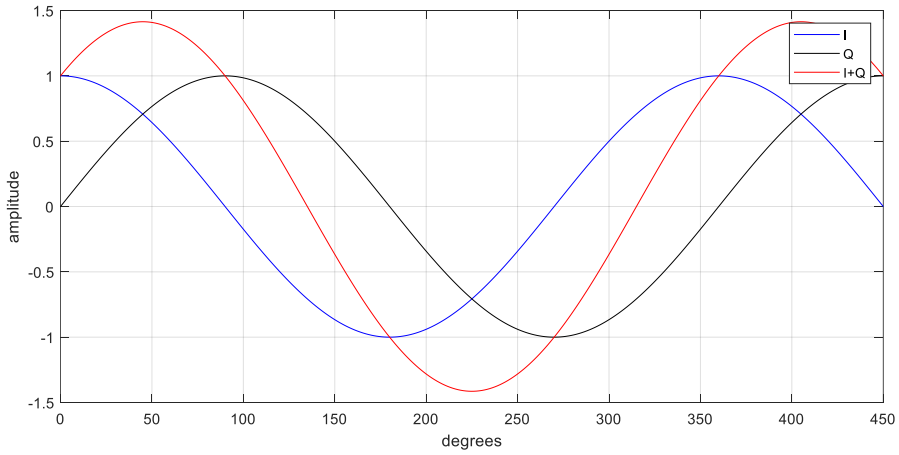


Figure 19. Adding the I and Q signals together

2.5.1.5 Modulation/Demodulation and IQ Signals

A digital signal is one whose amplitude takes on only a finite number of values. A digital signal whose amplitude takes on M values is an M-ary signal. Signals associated with a digital computer are digital because they can take on only two values, 0 and 1, a Binary or 2-ary signal. Digital Modulation begins with the formation of a digital message. A digital message then is fundamentally a sequence of binary 1s and 0s, derived for example from the byte patterns in a text or image file. These sequences of bytes are then modulated with an M-ary quadrature modulation scheme. Some commonly used quadrature modulation schemes are Binary Phase Shift Keying (BPSK), quadrature amplitude shift keying (4ASK), and Quadrature Phase Shift keying (QPSK), orthogonal quadrature phase shift keying (OQPSK), 8PSK, 16 Quadrature Amplitude Modulation (16QAM), 32QAM, 64QAM, Amplitude-Modulation Double-Sideband (AM-DSB), Amplitude-Modulation Single-Sideband (AM-SSB), Continuous-Phase Frequency-Shift-Keying (CPFSK), Gaussian Frequency-Shift-Keying (GFSK), Wideband Frequency Modulation (WBFM), and four-level Pulse-Amplitude Modulation (PAM4), and Quadrature Amplitude Modulation M-ary (QAM-M).

BPSK is commonly used for transmitting digital information wirelessly. It is defined as modifying the phase of the carrier $C(t) = \sin(2\pi ft)$ by $\theta(t)$ equal to 0- or 180-degree phase shifts, therefore yielding $C(t) = \sin(2\pi ft + \theta(t))$. Projecting information onto the waveform carrier can be implemented by setting the carrier phase to 0 degrees to represent a bit value of 0, while setting the phase to 180 degrees represents a bit value of 1. Figure 20 shows an example of a BPSK sine

carrier wave modulated with a binary message “01010”. Note how that phase shift occurs to the carrier when switching between the message bits 0 and 1.

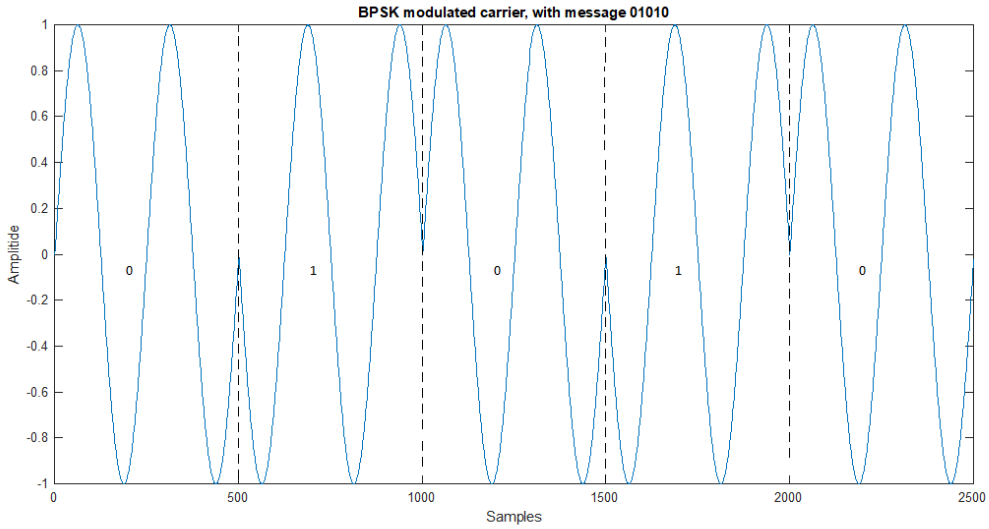


Figure 20. BPSK Modulated Carrier Message

For an ideal BPSK signal, the in-phase component I(t) with respect to is defined as $I(t) = \{+1, -1\}$, whereas the quadrature component Q(t) is always 0. It is common to view digitally modulated waveforms as phasors or in the complex IQ plane. In doing so, one can visualize the IQ pattern for a given waveform modulation type which in turn can allow for classification. For

example, an ideal noiseless BPSK signal would appear in a constellation image as shown in Figure 21.

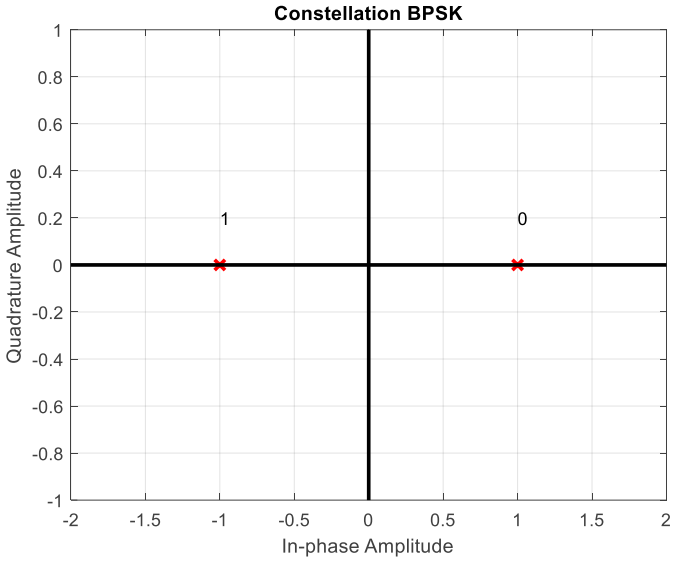


Figure 21. BPSK Constellation

Table 5 summarizes the BPSK IQ symbol table, with a digital-bit value of 0 represented by I = 1, and Q=0. A digital-bit value of 1 is represented by I = -1 and Q=0. Q is always held a 0.

Table 5. BPSK IQ Symbol Table

Symbol	I	Q
0	1	0
1	-1	0

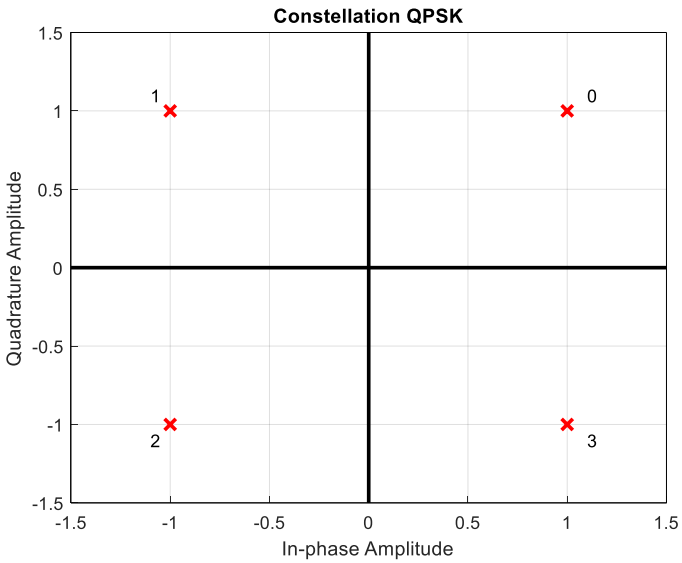


Figure 22. QPSK Constellation

As another example, QPSK could have a constellation that appears as in Figure 22. For QPSK, 2-bit symbols are used for data representations. Table 6 shows the IQ symbol table for QPSK. For example, I=1 and Q=1 represents a symbol value of 0.

Table 6. QPSK IQ Symbol Table

Symbol	I	Q
0	1	1
1	-1	1
2	-1	-1
3	1	-1

For classification purposes, one can clearly see from the constellation plots the differences between BPSK and QPSK. BPSK has only two states shown with red dots, while QPSK has four states. It is because of these unique patterns among the different modulation waveforms that has led to research in using convolutional neural networks for waveform classification.

For RF transmission, up-conversion must be performed where the I(t) and Q(t) components are first multiplied by a LO sinewave frequency and then added together as indicated in Figure 23. Multiplication is done to shift the baseband IQ signals to higher frequencies suitable for RF transmission. The addition is then performed to combine the up-converted I and Q components into a single RF waveform for transmission. Quadrature Modulation and Up-conversion is accomplished in combination with digital signal processing and/or with analog mixer hardware.

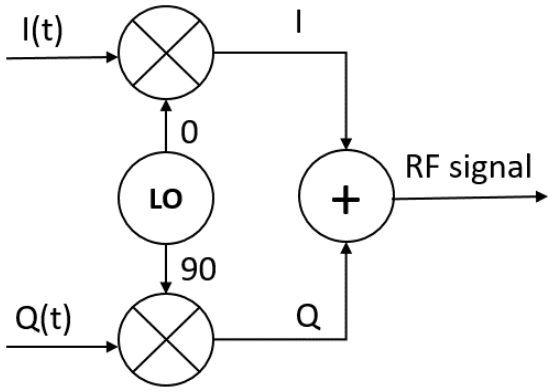


Figure 23. Transmitter - Quadrature Modulation and Up Conversion

To receive the transmitted signals, the Quadrature Demodulation and Down-Conversion process is carried out as shown in Figure 24. The RF-signal is split into two paths and then multiplied by a sine and cosine LO carrier. The LO frequency is assumed to be the exact frequency the RF signal was transmitted at. Then a low pass filter is applied to remove the unwanted higher order frequency components. The result is the I(t) and Q(t) quadrature components which can then be decoded to extract the symbols.

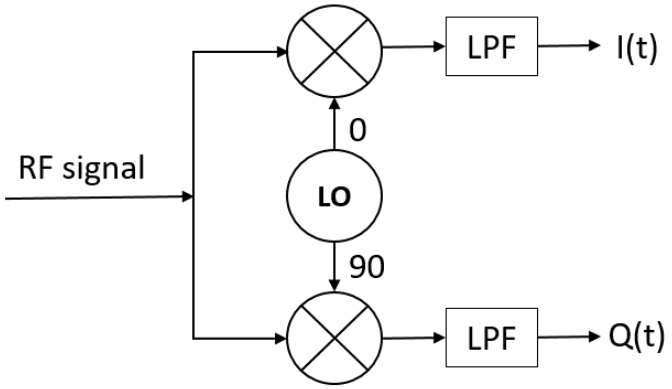


Figure 24. Receiver - Quadrature Demodulation and Down Conversion

In practice, to demodulate the received quadrature signals it is necessary for the receiver to know the carrier and symbol rate. When simulating the transmission and reception of waveforms, all parameters are known prior, and therefore demodulation is straight forward by using standard demodulation techniques. However, in practice when a receiver acquires a RF signal from a remote transmitter there are significant challenges that arise: the transmitter and receiver are disjoint systems and do not share the same LO clocks; clock frequency drift errors occur due to temperature effects; clock frequencies of two disjoint systems are never precisely equal; atmospheric conditions cause RF signal bounce and fading which can affect the received RF signal's amplitude and phase; and atmospheric noise can affect the signal to noise ratio of the received RF signal. All these factors impact the bit error-rate of received signals and must be dealt with.

2.5.1.6 Recovery of Symbol Sampling

Symbol timing recovery [93], is especially important for receiver demodulation of digital waveforms being recovered from a transmitter source. The requirement is to resample the input so there is one sample at the center of each symbol duration, and an integral sample-per-symbol rate.

This synchronization is achieved using digital phase-lock loops and resampling techniques such as polyphase resampling. The process of synchronization happens at three levels. First, when there is a carrier in the transmitted signal, a reference carrier is used by the receiver to create a baseband signal. Second, there is a need to perform bit synchronization where the receiver clock must synchronize with the symbols in the baseband signal. The third level includes word, frame, and packet synchronization.

As part of demodulating the carrier frequency acquisition, phase locked loops (PLL) can be used to track the phase and frequency of the incoming RF signal. Two commonly used PLL methods are the Squaring method and the Costas Loop briefly discussed below. The Classical Costas Loop shown in Figure 25 can be used for carrier recovery and signal demodulation of BPSK waveforms.

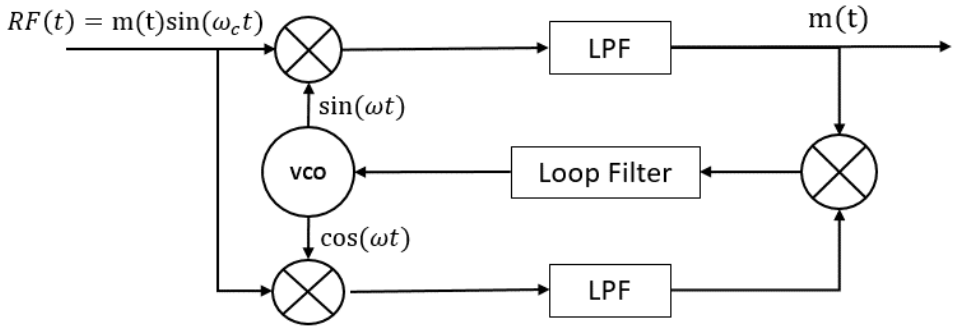


Figure 25. Classical Costas Loop [83]

$RF(t) = m(t)\sin(\omega_c t)$ is the RF signal received from the remote transmitter, where $m(t)$ is the BPSK message sent by the transmitter, and takes on the value of +/-1, and $\sin(\omega_c t)$ is the carrier frequency of the signal received from the remote transmitter. The quadrature voltage-controlled oscillator (VCO) uses the output of the narrow low pass feedback loop filter to allow the

VCO's ω frequency to lock to the fluctuating carrier frequency ω_c. When the frequencies are locked, the input to the low pass filters (LPF) is the result of multiplying RF(t) by the sine and cosine components. The LPFs are designed to pass the baseband signal which contains $m(t)$ [83]. In practice, the classical Costas Loop can be used to demodulate BPSK waveforms. As modulation schemes get more complicated, the demodulation algorithms become more complex. Also, to efficiently demodulate an RF waveform, it is best to have prior information such as the specific modulation scheme, the carrier frequencies, and the symbol rate of the waveform. Not knowing this information ahead of time will require a more complex demodulation scheme to succeed. It was of interest in this research to explore generating constellation images from arbitrary RF waveforms, however doing so is not a trivial task, and requires prior knowledge about the received waveform, or sophisticated demodulation techniques which is beyond the scope of this effort.

2.5.2 Spectrum Sensing

Spectrum sensing is defined as techniques employed by cognitive radio to monitor the RF for channel activity of the primary user to communicate on an unused channel. The detection theory is based on matched filtering, energy detection, radio identification sensing, and waveform-based sensing[17]. Spectrum sensing is a key component in establishing cognitive RF systems to allow them to know the state of their environment to make future decisions on how to respond to its environment.

2.5.2.1 Simple Energy Detection

Energy detection is the simplest method of spectrum sensing having low computational complexity. An energy detector can be modeled by considering the samples of a complex signal $x(n)$ to have zero mean and variance σ_x^2. The signal is assumed to be transmitted through a noisy medium with Additive White Gaussian Noise, $w(n)$. If no signal is transmitted the received signal meets the hypothesis $H_0: r(n) = w(n)$, with only noise. If a signal is transmitted through the noisy medium, the received signal meets the hypothesis $H_1: r(n) = x(n) + w(n)$.

The energy detector E is defined by summation of the product of the received signal $r(n)$ and its complex conjugate $r(n)^*$, $E = \sum_{n=0}^{N} r(n)r(n)^*$. If energy exceeds the threshold τ, the signal is declared present, otherwise it is declared not present as in

$$H_1: E > \tau, signal\ present, \text{ and}$$
$$H_0: E \leq \tau, no\ signal\ present\ .$$

Historically, an energy detector is the oldest and simplest detection technique that achieves good performance for reporting a bands occupancy when the SNR is strong [94], however, simple energy detection cannot detect signals with low SNR and does not allow for waveform classification [95].

2.5.2.2 Waveform Based Sensing

Known patterns are usually used in wireless systems to assist synchronization or for other purposes, for example patterns may include preambles, mid-ambles, regularly transmitted pilot patterns, or spreading sequences. The key point here is that the patterns are known. This approach

is also known as coherent processing. The performance of the sensing algorithm increases as the length of the known signal pattern increases[96].

Given a received signal $r(n) = x(n) + w(n)$, where $x(n)$ is the original transmitted signal, $w(n)$ is noise, then a decision metric $M = Re\,[\sum_{n=1}^{N} r(n)x^*(n)]$ where * is the conjugation operator, can be used to sense the presence of an expected signal x(n) by comparing *M* against a fixed threshold λ.

2.5.2.3 Cyclostationary Processing

To work around the limitations of power detection algorithms a more sophisticated approach called cyclostationary [94] processing has been investigated. Cyclostationary processing has the advantage of greater reliability at sensing low SNR signals than simple energy detection and can be used for classification of labeled data by training on known data sets. Cyclostationary detectors exploit hidden periodicities present in man-made signals, but absent from noise. Although more computationally expensive, the advantages of cyclostationary processing are it addresses blind spectrum sensing, where no features are known about a waveform [94]. Cyclostationary processing has been performed in the time and frequency domains, and presented by [20] is a frequency domain technique using cyclic spectral analysis for extracting the signal features, followed by an SVM classifier for the pattern recognition stage. Because cyclostationary processing is capable of blind spectrum sensing, using it for detecting new or unlabeled waveforms is a viable choice.

2.6 OVERVIEW OF RF SIGNAL CLASSIFIERS

A most common theme across cognitive architectures is the need to gain knowledge of the RF environment for decision making processes. Two major components of the cognitive RF architecture that must be automated to deal with real-time constraints, is to determine the occupancy state of the RF spectrum using sensing techniques, and to determine the RF modulation type occupying the spectrum by using waveform classification techniques. RF waveform classification is important for cognitive RF architectures because it can allow for tracking the behavior of specific emitter classes in covert and overt operations, and therefore can support military applications such as surveillance and electronic warfare, or can support the civilian sector for interference identification and spectrum management [21, 42].

Automatic modulation recognition methods can be grouped into two categories, where the first is a likelihood decision theoretic approach using hypothesis testing techniques, and the second is the feature-based pattern recognition which includes data acquisition, pre-processing, feature extraction and decision making by matching stimulus with stored patterns. The feature-based technique is considered most effective because it is more computationally efficient than the likelihood theoretic approach [21].

2.6.1 Feature Selection

The first step for applying an RF modulation classification technique is in selecting the appropriate features. The remainder of this section provides an overview of features that can be used in RF waveform classification. Features selected for RF waveform modulation classification

typically fall under waveform parameters, higher-order statistics on the signal and features extracted from the signal or other features.

2.6.1.1 Instantaneous Amplitude, Phase, and Frequency

To perform classification of RF modulated waveforms, Nandi [42] uses the three waveform parameters: instantaneous amplitude; instantaneous phase; and instantaneous frequency, to generate five key features to be used in decision tree structures to classify the FSK2, FSK4, ASK2, ASK4, PSK2 and PSK4 modulated waveforms. The five key features (further described in [42]) generated are: the maximum value of the spectral power density; the standard deviation of the absolute value of the non-linear component of the instantaneous phase; the standard deviation of the nonlinear component of the direct instantaneous phase; the standard deviation of the absolute value of the normalized-centered instantaneous signal amplitude; and the standard deviation of the absolute value of the normalized-centered instantaneous frequency. Although this work demonstrates satisfactory performance, the primary shortcoming is that it requires expert knowledge to define the features for specific waveforms.

2.6.1.2 Moments and Cumulants Higher Order Statistics (HOS)

To generate features for waveform classification, the HOS statistics cumulants have been shown to perform well in RF multipath fading channels [18, 19, 42], and a significant advantage of using them is that they can generate feature vectors from the raw data quickly with simple formulas. Cumulants are calculated from moments and the formulas shown in Table 7 is a summary of some

HOS recommended for digital waveform classification, with $M_{pq}\ and\ C_{pq}$ indicating the order of moments and cumulants.

Table 7. High order cumulants and high order moments proposed [18]

HOCs	HOMs Expressions
Second Order Cumulants	$C_{20} = M_{20}$ $C_{21} = M_{21}$
Fourth Order of Cumulants	$C_{40} = M_{40} - 3M_{20}^2$ $C_{41} = M_{42} - 3M_{20}M_{21}$ $C_{42} = M_{42} - \lvert M_{20}\rvert^2 + 2M_{21}^2$
Sixth Order Cumulants	$C_{60} = M_{60} - 15M_{20}M_{40} + 30M_{20}^3$ $C_{61} = M_{61} - 5M_{21}M_{40} - 10M_{20}^3M_{41} + 30M_{20}^2M_{21}$ $C_{62} = M_{62} - 6M_{20}M_{42} - 8M_{21}^3M_{41} - M_{22}M_{40} + 6M_{20}^2M_{22} + 24M_{21}^2M_{20}$ $C_{63} = M_{63} - 9M_{21}M_{42} + 12M_{21}^3 - 3M_{20}M_{43} - 3M_{22}M_{41} + 18M_{20}M_{21}M_{22}$

M_{pq} represents the moment of a complex signal defined as $M_{pq} = E[y(k)^{p-q}\ (y^*(k))^q]$. These features can be used to generate waveform features to train waveform classifiers such as decision trees, neural networks, and support vector machines.

2.6.1.3 Extracted Features

Feature extraction [97] is the process of using existing feature parameters to comprise a lower-dimensional feature space, and to map useful information contained by original features to a smaller number of features, ignoring redundant and irrelevant information or referred to as lower dimensionality reduction. Some of the most used feature extraction techniques are Correlation Analysis, Principal Component Analysis, Rough Set Theory, Fisher Discriminant Analysis, Kernel Transformations, Matrix Singular Value Decomposition, Statistical Uncorrelated Discriminant Analysis, Independent Component Analysis, and Projection Pursuit. The extracted feature set constitutes the features used for training the classifier.

2.6.2 Automatic Modulation Classification

Automatic modulation classification identifies the modulation type of a transmitted waveform from the received data samples. This section provides discussions on automatic modulation classification algorithms.

2.6.2.1 SVM Based Classification

Freitas et al [20] presents two approaches to the problem of automatic modulation classification in cognitive radios. One approach classifies QAM signals with a SVM classifier that uses the distance between the received symbol and its nearest neighbor in constellations as input parameters. The second approach uses cyclic spectral analysis for extracting the signal features, and the SVM classifier is used for the pattern recognition stage, achieving excellent results for AM, BPSK, QPSK and BFSK modulation schemes. However, both approaches do not address anomaly detection and online adaptation.

2.6.2.2 Decision Tree Based Classification

A set of criteria using decision trees for identifying different types of digital modulation is applied [21, 42]. The use of decision trees provides a fast and reliable technique to identify digitally modulated signals, where the waveform signature features used in the identification algorithm are calculated using the conventional signal processing methods and HOS.

2.6.2.3 Adaptive CNN Ensemble

An approach for online adaptive multispectral image classification in nonstationary environments is presented by [98], where an adaptive CNN ensemble includes multiple custom

pretrained CNN modules called dynamic ensemble classifiers (DEC) that can be created and updated while online, and uses a weighted voting ensemble approach for final classification. The number of CNN DEC modules increases dynamically after the detection and storage of new spectral band image arrivals into a sample repository module. After fifty images are stored, they are declared as new, and then the k-means algorithm is used (with K=1) to cluster the images, with the resultant cluster used to train a new DEC module while online.

2.6.2.4 CNNs for Modulation Recognition

The current understanding of the human vision system has inspired much research in the development of CNNs which perform image classification with accuracies surpassing that of humans. CNNs became of significant importance due to their high accuracy in image classification, particularly when AlexNet was created for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010) competitions. Sparked by the success of AlexNet, researchers have developed several CNN architectures, resulting in top-5 accuracy levels of identifying objects from 1000 categories reaching 95% in the ILSVRC[99].

Deep learning achieves state-of-the-art performance in image classification tasks. Modern models make use of deep CNNs such as AlexNet. However, training such models requires substantial amounts of data and training on large data sets can take tremendous time. In some situations where data is sparse, the number of training samples per class may be limited. A common approach to overcome this problem is to use transfer learning. The principle underlying transfer learning is to use knowledge learned on a source task and transfer that to a new problem space. It provides considerable benefits over learning from scratch, with a major advantage that a model can

learn more efficiently since it starts with a weight matrix [50] that is pre initialized to extract features that the new problem space has in common with the source problem space.

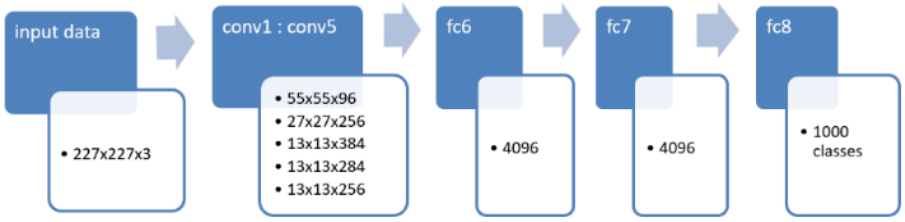


Figure 26. AlexNet - Simplified block diagram

A highly simplified block diagram of the original AlexNet CNN is shown in Figure 26. ***AlexNet - Simplified block diagram***. The input layer takes data inputs as two dimensional RGB-images of various objects with pixel dimensions 227-by-227-by-3. The input layer is followed by five convolutional layers conv1 through conv5, and finally two fully connected layers fc7 and fc8. The final fully connected layer (fc8) contains 1000 nodes with the node outputting the highest value indicating the selected class of the input image. The CNN of this type are originally trained on image objects such as dogs, cats, and cars. To take advantage of their high accuracy and by using transfer learning techniques, common approaches modify the pre-trained CNN to implement new classes of CNN classifiers.

This is typically done by replacing the last fully connected layer of the CNN with a new fully connected layer with the number of output nodes matching the number of new target domain of classes to be trained on. For example, as shown in Figure 26. ***AlexNet - Simplified block diagram***, the number of output classes in the network could be changed to 100 instead of 1000, and then the new layers of the network would be retrained on the new data set containing 100 classes. High accuracy levels can be obtained using this approach.

The technique described in detail by Peng et al. [22, 100], uses constellation plots to map the complex real and imaginary components of digitally modulated waveforms to generate 3-Channel images. Then, using transfer learning, the 3-Channel images are used to train CNNs as waveform classifiers. Figure 27 shows sample images generated for nine digital waveform types, and SNR levels ranging from 14dB, 4 dB, and -6 dB [101].

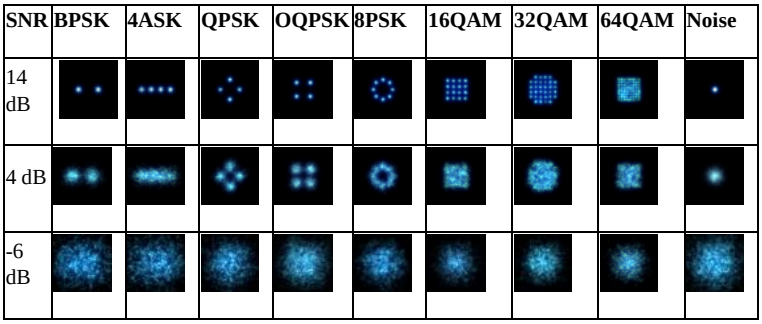


Figure 27. Sample of 3-Channel known images

The complex values are mapped to the discrete pixel positions of an image using the standard bilinear interpolation zoom technique. For example, the 3-by-3 constellations are mapped into a 277-by-277-pixel image using interpolation. The CNN inputs are then 3 channel RGB images and different information is conveyed on each channel. Since some complex values may map to the same pixel location, and it is desirable to maintain as much information as possible during mapping, the location of each sample pixel and the impact each sample pixel has on its neighboring pixels make use of an exponential decay model.

Hasanpour [102] demonstrated that although CNNs such as AlexNet, ResNet, GoogleNet and Inception have had significant impact on performance accuracy for classification problems, there is a significant need to reduce the millions of parameters these networks demand to minimize

computational complexity, particularly for embedded systems. They have shown that a simple 13-layer architecture outperforms most of the deeper and complex architectures.

Research by West and O'Shea [49], [103] has shown that Deep Neural Networks (DNN) are pushing performance boundaries of machine learning tasks in the field of cognitive radios with some successes showing that simple convolutional neural networks outperform algorithms with decades of expert feature searches.

Despite the dramatic breakthrough in performance, CNNs are brittle in that once architected and trained they can only perform on the class of data in which they were trained. Therefore, there is a need to change the processing behavior to maintain performance in changing environments that the CNNs were not trained on.

3 CLASSIFICATION OF RADIO FREQUENCY DATA STREAMS WITH CNN TRANSFER LEARNING

In this chapter we describe some of the challenges associated with radio waveform classification, and then our methodology of baseband waveform generation, 3-Channel image formation, and CNN transfer learning technique used for the training and classification of the waveforms. We also describe the datasets used to train and test, and finally we discuss the results. An important outcome of this effort is the establishment of a framework for generating acceptable feature vectors suitable for input to our adaptive classifier algorithms.

3.1 CHALLENGES

In practice, RF waveform classification schemes present significant challenges. In many practical cases the mere detection of a waveform's presence is not trivial. Modulations such as spread spectrum or frequency hopping are inherently covert therefore making them difficult to detect. Even if modulation schemes are easily detectable, they are exposed to destructive transformations such as RF phase shifts due to man-made structures or ground bounce, or from fading effects due to atmospheric conditions. Furthermore, when using wide band receivers with bandwidths significantly spanning beyond the signal of interest's bandwidth, the received signals will inevitably consist of multiple independently transmitted signals that must be separated before applying classification algorithms. Finally, the signals received have varying levels of SNR levels which can significantly impact performance. For the dataset used in this research, we assume that the negative effects described are negligible, and that the RF signals have been sensed, isolated, down converted to baseband, and input to our classifiers.

3.2 CONTRIBUTIONS

This work uses transfer learning to extract features from images generated using the 3-Image technique from the works of Peng et al. [22, 100] to generate representative feature vectors for our adaptive classification work. The complex real and imaginary components of the digitally modulated waveforms are mapped to the RGB constellation images referred to as the 3-Channel images. The 3-Channel images are used as input to the CNNs for classification. Due to the assumption of training on limited data sets, transfer learning is a significant part of this effort to take advantage of the knowledge of previously trained CNNs, and to minimize training time with limited computer resources. This research investigates the performance among five different CNNs for RF waveform modulation classification. The results show a comparative analysis of the RF classification performance of the utilized CNNs with transfer learning. The data output from the CNNs is repurposed and used for generalized feature extraction for waveform classification and anomaly detection in later experiments. In taking this approach it is not necessary to explore in significant detail what feature vector is required as input to the adaptive classifiers, but instead to rely on selecting a representative feature vector from one of the layers of the pre-trained CNN.

3.3 METHODOLOGY

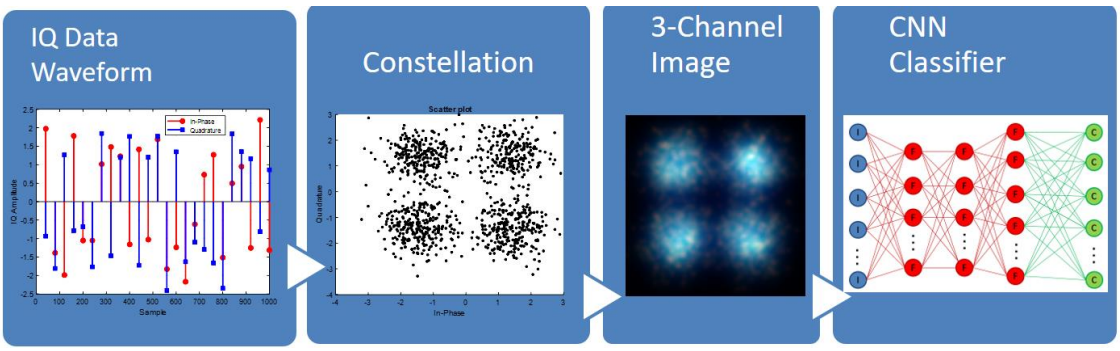


Figure 28. Methodology overview of RF classification

Figure 28 shows an overview of the methodology used for RF classification. Starting on the left, the continuous amplitude In-Phase and Quadrature (IQ) waveform is generated from a random 1000 symbol message. With the IQ data viewed as a constellation plot, the different modulation types will exhibit recognizable image patterns. For example, a noisy QPSK waveform is shown in the constellation with four blobs, with each blob indicating one of four possible symbols in the message. Because the IQ values are presumed to be from the continuous time domain, they must be mapped into discrete pixel dimensions when forming the 3-Channel image. From a dataset of random symbol messages, a dataset of 3-Channel images is generated and then transfer learning techniques are used for training and prediction assessments of the CNN waveform classifiers. The following sections discuss in more detail this methodology.

3.3.1 Waveform Generation

This section discusses in detail how the waveforms were synthesized. In practice, before RF systems transmit their RF carrier waveforms, they are modulated with the information (symbol)

bearing baseband signal at an RF bandwidth proportional to the symbol rate of transmission. During RF propagation from the transmitter to the receiver, the waveform is degraded as it is exposed to terrestrial and atmospheric conditions, and at the point of reception the waveform is demodulated to extract the baseband signal and to decode it. To simplify our approach, our methodology assumes that the waveforms have been modulated, transmitted, received, and demodulated to baseband, and it is the baseband signal we process during this experiment. The remainder of this section describes in detail how the baseband signals are generated and utilized for the experiment.

To generate a baseband signal, a message stream $m(t)$ containing the desired information must first be defined. We generate a uniformly distributed random message stream of symbols $m(t)$ where $t = 1 \dots N$, and N defines the message length. With M representing the number of symbol states of $m(t)$, the valid symbol values are defined as $s \in \{0,1, \dots M-1\}$. As an example, if $M = 2$, then $s \in \{0,1\}$, and a valid message stream of length N=4 could be m(1)=1, m(2)=0, m(3)=1, m(4)=1. Each symbol within the message stream is then mapped into the IQ constellation space, $C_s = A_s^I + jA_s^Q$ where A_s^I and A_s^Q are the in-phase and quadrature mapping values for the symbol s. This will result in the message stream $m(t)$ being mapped into the complex baseband signal $C(t)$.

The type of waveform class is dependent on the selected mapping function. The 9 modulation classes generated for this experiment BPSK, 4ASK), QPSK, OQPSK, 8PSK, 16QAM, 32QAM, 64QAM, and Noise. Further details of the modulation mapping functions are discussed in the remaining subsections.

3.3.1.1 B-PSK Mapping

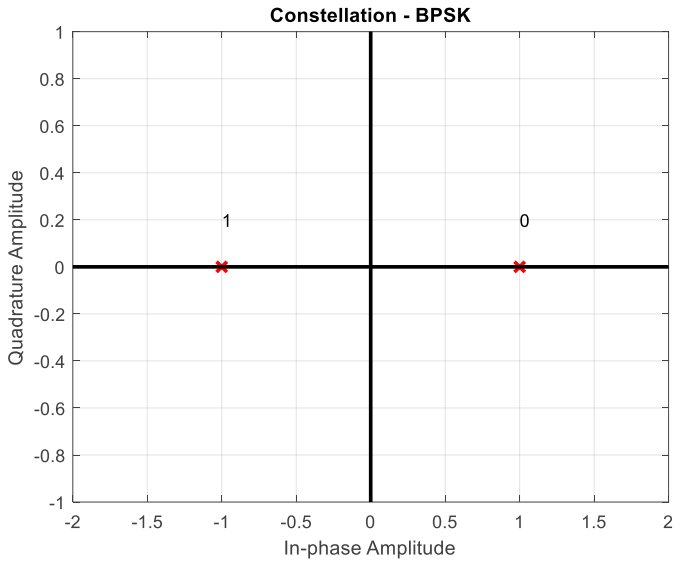


Figure 29. Ideal BPSK

Figure 29 shows an ideal BPSK signal with only two (2) possible symbols, $s \in \{0,1\}$. Symbol 0 falls on the in-phase axis at position 1 and represents a binary 0, and symbol 1 falls on the in-phase axis at position -1 and represents a binary 1. This yields phase shifts 0 and π in the complex plane. This mapping is summarized by $C_s = A_s^I + jA_s^Q$: $\{A_0^I$=1, A_1^I=-1, $A_s^Q = 0\}$, where $C_0 = +1$, yields a binary 0, and $C_1 = -1$ yields binary 1.

3.3.1.2 M-PSK Mapping

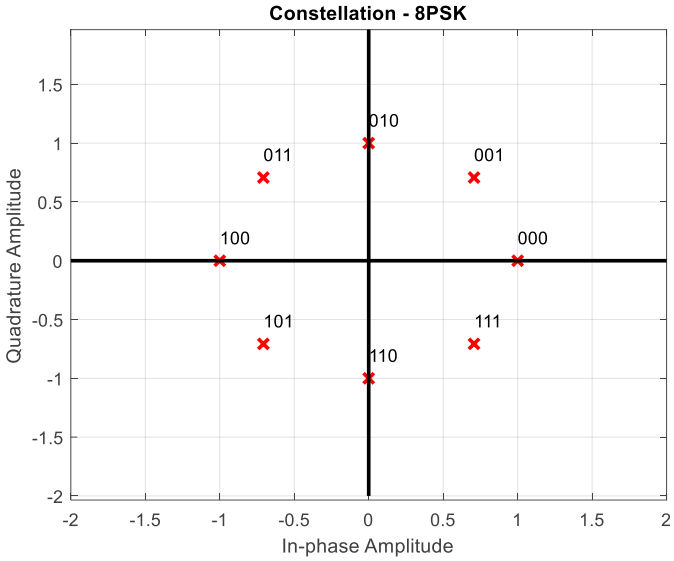


Figure 30. Ideal 8PSK

The M-PSK complex baseband signal C_s is generated by properly choosing the polar values r and θ_s in the complex plane of $C_s = r\angle\theta_s$. r is a fixed scalar representing the amplitude in the complex space, and θ_s is the angle in radians. Given M as the number of unique symbols, Figure 30 shows an example with M=8, which is the ideal 8PSK constellation with eight (8) possible symbol states. Each symbol value determines the angle on the complex plane. The equations

$$C_s = (r, \theta) : r = c, \theta \in \left\{\frac{\pi}{M}, \frac{3\pi}{M}, \frac{5\pi}{M}, \dots, \frac{(2M-1)\pi}{M}\right\},$$

$$M = \{2,4,16,32,64 \dots\}, \text{ and}$$

$$s = \{0, 1, 2, \dots, M-1\},$$

summarize the parameters for M-PSK modulation. For experimentation, M was set to 4 and 8 for QPSK(4PSK) and 8PSK respectively, and the assignment of a unique symbol to a corresponding phase angle was arbitrary.

3.3.1.3 M-QAM Mapping

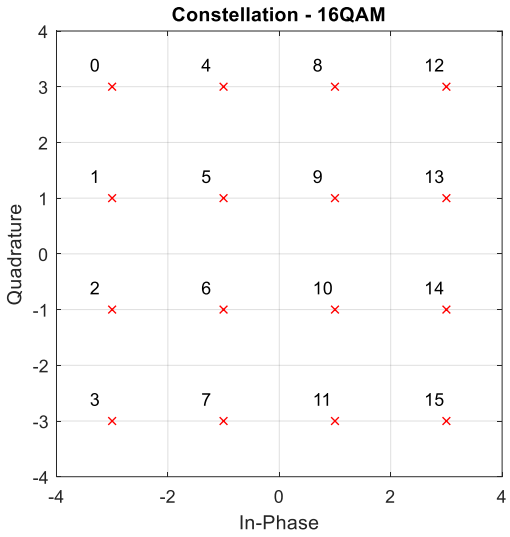


Figure 31. Ideal 16QAM

The M-QAM complex baseband signal C_s is generated by properly choosing $C_s = \left(A_s^x, A_s^y\right)$ such that the conditions of equation 3 below are met. Given M as the number of unique symbols, Figure 31 shows an example where M=16, which is the ideal 16QAM constellation representing sixteen (16) possible symbol states. Each symbol value determines the $\left(A_s^I, A_s^Q\right)$ pair in the complex plane. The equations

$$C_s = \left(A_s^I, A_s^Q\right) : \; A_s^I, A_s^Q \in \{+/-1, +/-3, +/-5 \ldots, +/-\sqrt{M}-1\},$$

$$M = \{4,16,32,64, \ldots\}, \text{ and}$$

$$s = 0,1,2, \ldots, M-1$$

summarizes the M-QAM mapping function. In this experimentation M was set to 16, 32, and 64 for 16QAM, 32QAM, and 64QAM, respectively, and the assignment of a unique symbol to a corresponding phase angle was arbitrary.

3.3.1.4 4ASK Mapping

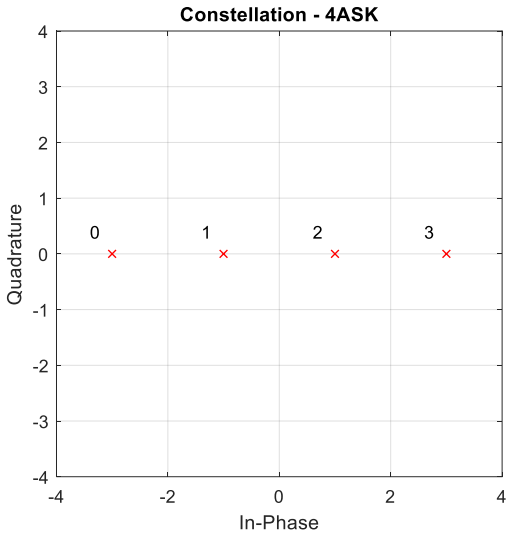


Figure 32. Ideal 4ASK

The 4ASK baseband signal C_s is generated by choosing the values A_s^I, A_s^Q such that the conditions

$$C_s = \left(A_s^I, A_s^Q\right) : \; A_s^Q = 0, \; A_s^I \in \{+/-1, +/-3\},$$
$$M = 4, \text{ and}$$
$$s = 0,1,2,3$$

are met. Figure 32 shows a constellation example of an ideal 4ASK waveform where the number of valid states are four (4).

3.3.1.5 Additive White Gaussian Noise (AWGN)

After the complex baseband waveform signals $C(t)$ are synthesized, they are summed with AWGN $n(t)$ to simulate transmission noise effects, resulting in $C_{AWGN}(t) = C(t) + n(t)$. For experimentation we added varying levels of AWGN per equation $SNR_{dB} = 20log_{10}\left(\frac{RMS\{C(t)\}}{RMS\{n(t)\}}\right)$,

where $RMS\{\vec{x}\} = \sqrt{\frac{1}{N}\sum_n x^2[n]}$. The SNR levels were varied from -6, -5, … 14 dB.

3.3.2 3-Channel Image Generation

Using the approach by Peng et al [22, 100], we form 3-Channel images of the modulated baseband complex signals $C_{AWGN}(t)$ with signal to noise ratios varying from -6dB to 14dB, yielding typical complex plane values ranging from +/-3. For simplicity, any values exceeding +/-3 were excised from processing. Samples of the 3-Channel images are shown in ***Figure 33***.

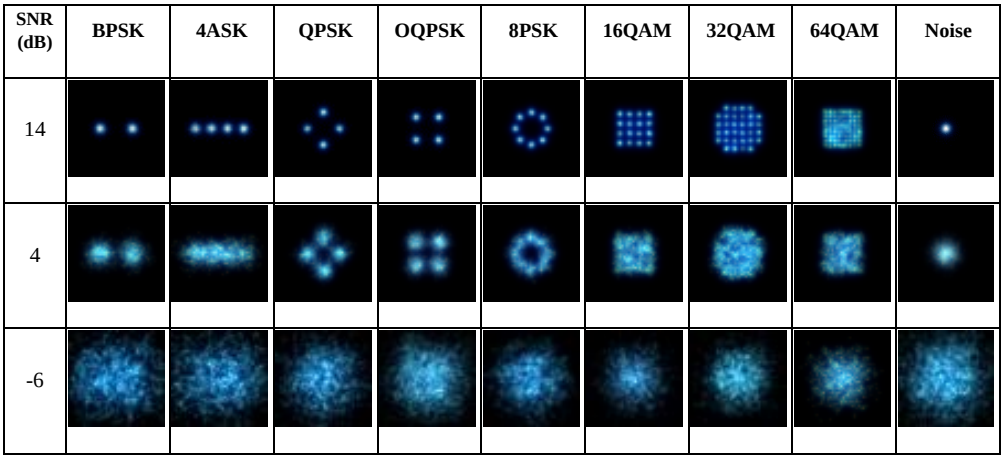


Figure 33. Samples of 3-Channel known waveform training images

The complex continuous values of $C_{AWGN}(t)$ are mapped into discrete N-by-N-by-3-pixel images using the standard image processing bilinear interpolation method [36]. For the 3-Channel RGB images the technique conveys different information on each channel. Since some of the complex values may map to the same pixel location, and it is desirable to maintain as much information as possible during mapping, the location of each sample pixel and the impact each sample pixel has on its neighboring pixels make use of an exponential decay model [22, 100]. The decay model is defined by $B_{ij} = Pe^{-\lambda d_{ij}}$ and $I_j = \sum_{i=0}^{n} B_{ij}$, where B_{ij} is the impact of sample

point #i on pixel #j, and I_j is the calculated intensity of pixel #j defined as the decayed summation over all samples in B_{ij}. P is an intensity constant set equal to 1, d_{ij} is the distance between sample point #i mapped to the image space and the centroid of pixel #j, and finally λ is the exponential decay rate for each RGB channel.

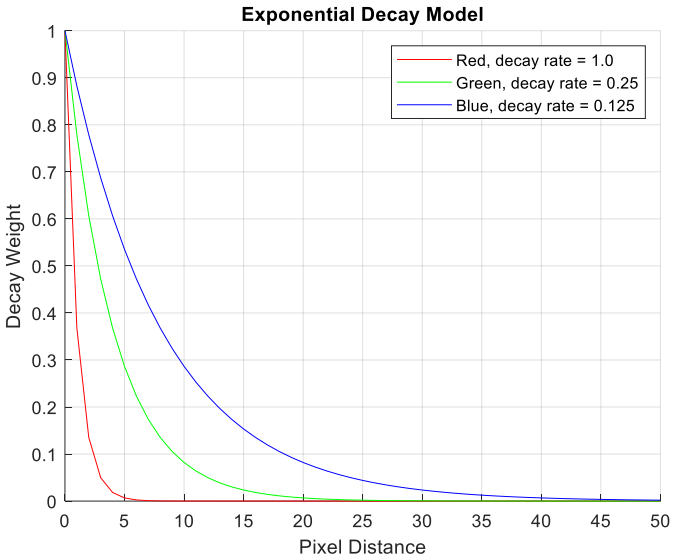


Figure 34. 3-Channel exponential decay model

To retain as much information as possible when mapping, the three different exponential decay rates were applied to the complex samples, with Figure 34 showing the behaviors of the 3-channel exponential decay models as function of decay weights and the pixel distances. The slower the decay rate, the greater the impact that distant IQ samples have on the IQ sample position under evaluation. From experimentation, the selected decay rates for the RGB channels are $\lambda_{R=1,}$ $\lambda_{G=1/4,}$ $\lambda_{B=1/8}$. Since the blue RGB channel has the slowest decay rate, a blue-hue dominates the 3-Channel image colors.

3.3.3 CNN Transfer Learning

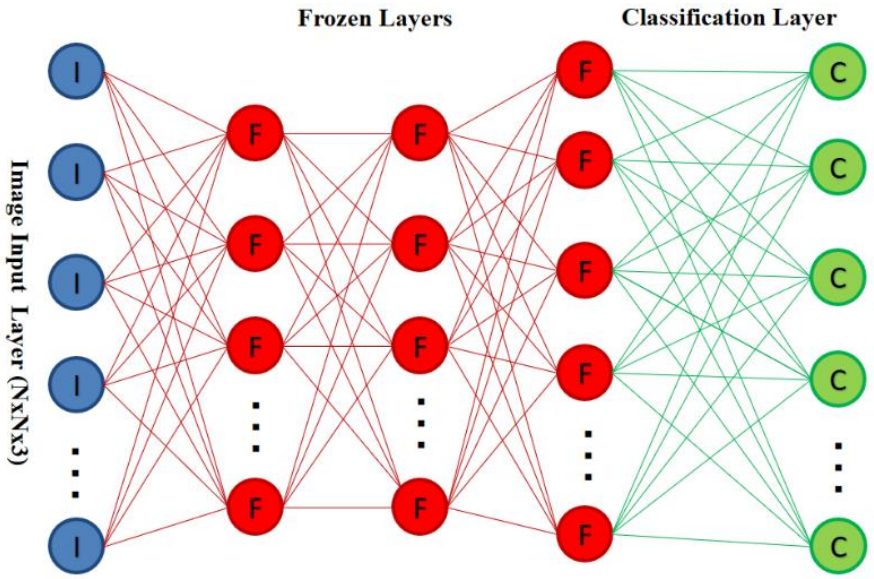


Figure 35. Simplified CNN Architecture

We used five popular CNN image classification networks: AlexNet; GoogleNet; Inception V3; ResNet50; and MobileNet. The networks have been demonstrated in literature as highly accurate classifiers and provide the opportunity to utilize transfer learning by leveraging the general knowledge the classifiers possess from prior training on millions of images. Figure 35 depicts a highly simplified diagram of the CNN architectures. Each CNN has a three (3) dimensional image input layer *(annotated by the blue nodes marked I)* having row by column by channel pixel dimensions of NxNx3, where the value of integer N is specific to the CNN, and the value 3 represents the three RGB color channels. The image input layer feeds into the frozen hidden layers *(annotated by the red lines and nodes marked F)* composed of multiple convolutional and other layer types which are specific to the chosen network. Making use of transfer learning techniques, the weights and biases parameters of the frozen layers are initialized to those obtained from the original CNNs pre-trained on ImageNet data, and the assigned parameters do not change during training. The last classification layer *(annotated by the green*

lines and nodes marked C) is a fully connected Softmax layer used to predict the class of the input image.

Table 8. CNN Parameters

CNN Name	Input Layer Pixel Dimensions (NxNx3)	Learnable Parameters (Million)	Operations for Single Forward Pass (G-ops)	Feature Vector to Classification Layer (Length F)	Original # of Classes
AlexNet	227x227x3	60	2.5G	4096	1000
GoogleNet	224x224x3	7	3 G	1024	1000
InceptionV3	299x299x3	24	12G	2048	1000
ResNet50	299x299x3	25	8 G	2048	1000
MobileNetV2	224x224x3	3.5	0.3G	1280	1000

Shown in Table 8 are CNN parameters to further describe the methodology applied using the CNN architectures. Column one (1) provides the name of each CNN. Column two (2) defines the image input layer's pixel dimensions for the CNN, and one can observe that the required resolutions are relatively low in comparison to standard high-definition images (such as 1920x1080 pixels). The 3-Channel images generated from the continuous waveform data must be interpolated to these discrete pixel sizes before being input into the CNNs. Column three (3) has the number of learnable parameters for each CNN where the values account for the total number of weights and biases for the original network. The learnable parameters give a relative indication of CNN training time. Column four (4) shows the number of operations for a forward single prediction pass and is based on the common practice of considering the number of multiplication and addition operations, and the number of operations is indicative of the CNN computational speed. Column five (5) provides the length of the feature vector feeding into the Softmax classification layer. The feature vector length is the number of activation outputs of the last frozen layer. For example, AlexNet has a 4096-length feature vector that is input into the last classification layer. Column six (6) shows the original number of image classes the networks originally trained on. All CNNs were initially trained on the same datasets consisting of 1000 classes. For our experiments, it is this last classification layer that will be removed and replaced with a new layer having the number of Softmax nodes

equal to the number of RF waveform classes in our datasets, and with a newly trained set of weights and biases for each class.

Using transfer learning we take advantage of the low training dataset requirements, and the high-performance accuracy that can be obtained with minimal training using previously trained CNNs. We modified the pre-trained CNNs to implement feature extraction for RF waveform modulation classifiers. For this experiment, we froze the original weights and biases of all CNN layers except the last classification layer. We then replaced the 1000-node classification layer with a 9-class output layer where the weights and biases of this new layer were backpropagation trained on the RF waveform datasets. The average of the input images from the training dataset is calculated and is used to normalize input images during predictions [104].

3.4 DATASETS

Table 9 summarizes the dataset for this experiment. The datasets were generated as described in the methodology section. The datasets consisted of the modulation types BPSK, 4ASK, QPSK, OQPSK, 8PSK, 16QAM, 32QAM, 64QAM, and Noise. For each modulation type, 1050 samples were generated with varying levels of SNR. The SNR levels range from -6dB to 14dB. SNR levels were generated in 1 dB increments resulting in 50 samples at each SNR level for each modulation type. For training and testing, 700 and 350 samples were respectively taken from each modulation sample set. Each sample is converted to a labeled 3-Channel image for processing.

Table 9. Dataset summary (knowns)

Modulation Type	Total Samples		Symbols per Sample	SNR Min	SNR Max	SNR Step	#SNR Levels	Samples Per SNR	Training Samples	Test Samples
BPSK	1050		1000	-6	14	1	21	50	750	300
4ASK	1050		1000	-6	14	1	21	50	750	300
QPSK	1050		1000	-6	14	1	21	50	750	300
OQPSK	1050		1000	-6	14	1	21	50	750	300
8PSK	1050		1000	-6	14	1	21	50	750	300
16QAM	1050		1000	-6	14	1	21	50	750	300
32QAM	1050		1000	-6	14	1	21	50	750	300
64QAM	1050		1000	-6	14	1	21	50	750	300
Noise	1050		1000	-6	14	1	21	50	750	300

3.5 EVALUATION

Table 10. 9-Class CNN Classifiers, Results from Test Data

CNN	Top-1 % Accuracy SNR: -6 to 14 dB	Top-1 Accuracy SNR: 4 to 14 dB	% Difference High vs. Low SNR
Google Net	87	98	11
MobileNet	87	98	11
Inception V3	86	98	12
Alex Net	84	96	12
ResNet 50	82	93	11

Table 10 shows the overall accuracy results from the test data of the 9-class CNN modulation classifiers. Column two shows the Top-1 % accuracy of the five CNN waveform classifiers with lower SNR data sets. GoogleNet, MobileNet, and Inception V3 provide the greatest performance, whereas ResNet50 provides the least performance. Column three shows the Top-1 % accuracy results of the CNN waveform classifiers with the higher SNR datasets. In this case, MobileNet, GoogleNet and Inception V3 again are the top performers, and ResNet50 provides the least performance. For each CNN, there is a performance degradation of 11-12 % when classifying the high verses low SNR datasets.

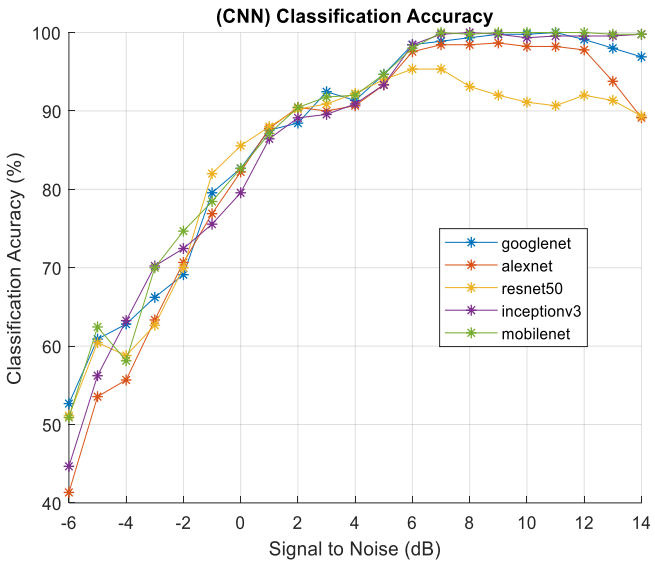


Figure 36. CNN Classification verses SNR

Figure 36 shows the CNN classifier results as a function of SNR. As one might expect, the accuracy of performance increases as SNR increases. ResNet50, AlexNet, and GoogleNet show unexpected performance drops for high SNRs between 6 to 14 dB. It is surmised that the performance drops can be resolved with further training effort, and by considering the possibilities of over-fitting or under-training on the training datasets. Further training is not pursued because the primary goal is not to establish optimal performance of the CNNs, but to establish networks that provide some reasonable level of performance for comparisons purposes.

3.6 SUMMARY

We developed a detailed framework for generating modulated waveforms. We then described an approach for mapping modulated baseband waveforms into 3-Channel images. We then explored using transfer learning on five CNNs for waveform modulation classification. Overall, the CNN waveform classifiers perform well with the higher SNR levels, however the

performance will degrade if the classifiers are presented with data classes that they were not trained on, and therein lies the problem with non-adaptive classifiers operating in non-stationary environments.

To tackle this problem, we turn our attention to exploring adaptive anomaly detection techniques to isolate known waveforms from new waveforms before they are presented to the pretrained classifiers. In the remaining sections we will use this overall framework for feature generation for the development of anomaly detection and adaptive waveform classification.

4 PCA ANOMALY DETECTOR

To perform anomaly detection in a multivariate population one common approach is to model the population with a multivariate normal distribution, and to use a distance measure to decide when a random sample falls outside of expected threshold boundaries it would be declared an anomaly. However, for high-dimensional data, the large number of features results in a high computational burden, and therefore it may be more feasible to map the original features into a lower dimensional space, and then carry out the anomaly detection algorithm[105].

This section describes preliminary work with the primary goal of demonstrating that monitoring the high dimensional last layer of transfer learned CNNs with a normal distribution in PCA space can reliably detect anomalous waveforms. This effort is explorative in assessing the separability of the known versus the anomalous datasets and the results can be used as a baseline to gauge the performance of the adaptive anomaly detection algorithms developed under this research.

4.1 CHALLENGES

To achieve high levels of accuracy of RF waveform classification in non-stationary environments, it is necessary to possess anomaly detection capabilities to separate anomalous stimuli from otherwise presumed labeled data. Incorporating such a mechanism will improve the overall performance of a supervised learned classifier system. In taking this approach the feature vector space must lend itself to supporting anomaly detection within the same feature space as the known classes. Once an anomalous stimulus is flagged as such, it can be rejected as input to the fixed CNN classifier trained on labeled data.

4.2 CONTRIBUTIONS

The contribution of this work demonstrates that applying PCA on features extracted using transfer learning techniques with publicly available CNN architectures trained on massive datasets can reliably separate anomalous stimuli from labeled waveforms. This part of the study also provides baseline performance measurements for comparative analysis of the adaptive classifiers and anomaly detection methods derived in later sections.

4.3 METHODOLOGY

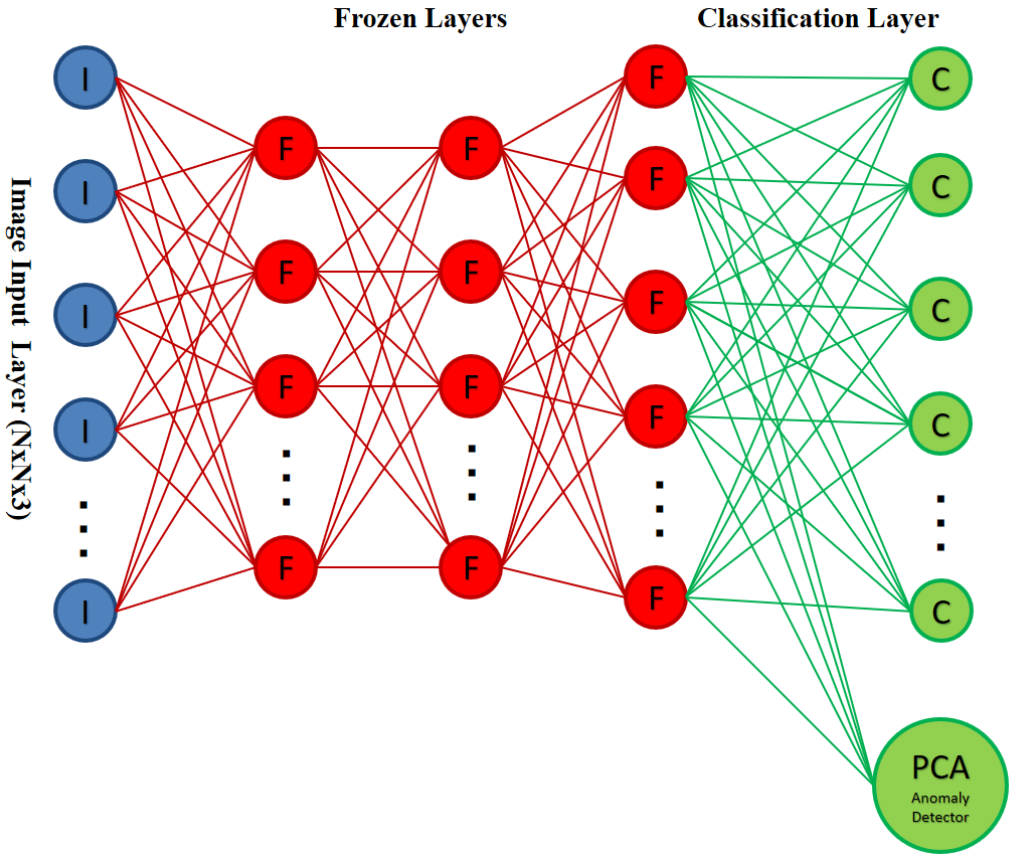


Figure 37. PCA Anomaly Detector in CNN

This approach assumes a normal probability distribution model using a normal distribution in PCA space to detect anomalous stimuli presented to the classifier. Figure 37. ***PCA Anomaly Detector in CNN*** shows the architecture where the PCA anomaly detector is attached to the last activation layer of the CNN, the same layer where classification is performed. After carrying out CNN training as described in section 3.3.3, when a stimulus is input to the modified CNN, the output value of the PCA anomaly detector is monitored, whereby if the value falls outside predefined thresholds, the input stimulus is declared an anomaly and rejected for classification, otherwise it will be classified according to one of the currently known classes. The remainder of this section will discuss in detail the implementation of this approach with particular emphasis on the anomaly detector.

Given input stimulus x, a k dimensional column vector, then Σ is the covariance matrix of the feature space of x, μ is the mean vector of the features, Σ^{-1} is the inverse of the covariance matrix, and $|\Sigma|$ is the determinant of the covariance matrix. In this discussion the x feature vector is taken as the output of the activation function in the last layer of the frozen portion of the CNN as shown in Figure 37. ***PCA Anomaly Detector in CNN***. Then the expression

$$D(x) = (x - \mu)^T \Sigma^{-1} (x - \mu) \qquad (1)$$

is the Mahalanobis distance providing a separation measurement between the vector x and a normal distribution $\mathcal{N}(\mu, \Sigma)$[25]. We then define a null and alternate hypothesis H_o and H_1, where H_o states there is no anomaly and H_1 states there is an anomaly. We use $D(x)$ as our prediction measure, where there is a range of $D(x)$ real values, and thresholds $dmin$ and $dmax$ such that the null and alternate hypotheses are

$$H_o : dmin \leq D(x) \leq dmax, \;\; x \in \mathcal{N}(\mu, \Sigma) \text{ and}$$

$$H_1 : \; D(x) < dmin \; or \; D(x) > dmax, \;\; x \notin \mathcal{N}(\mu, \Sigma).$$

An anomaly detector can is implemented by monitoring the output activations of any layer of a CNN architecture, and we refer to such activations as a feature vector. In our study, we focused on monitoring the last fully connected CNN layer. For a given input to the network, when equation (3) holds, the input is declared an anomaly. Training the algorithm directly on the activation datasets yields a large feature space, leading to large covariance matrices, and therefore burdensome computations. For example, AlexNet has a 4096-activation feature vector x yielding a 4096-by-4096 inverse covariance matrix to compute $D(x)$. This will yield 4096^2 = 16777216 operations per x input vector. In general, to multiply a 1-by-n feature vector x by an n-by-n inverse covariance matrix, and then by the transpose of the feature vector matrix yields $O(n^3)$ complexity.

As is common practice, to reduce the number of features and the computational load, PCA is utilized to project the higher dimensional feature vector x and the covariance matrix, into a lower dimensional feature space. To implement, the training data set is a matrix X of size m-by-n, where m is the number of training sample vectors, and n is the number of activations or features per vector. If required, the matrix X can be preprocessed with a normalization transformation, $ntransform(X)$, to make the dataset as close to a normal distribution as possible, yielding an m-by-n matrix X_t. X_t is then processed by the PCA algorithm and a subset of $\mathbb{C}$ eigenvectors is chosen such that they have the greatest eigenvalues, $\mathbb{C} << n$.

Once $\mathbb{C}$ principal components are selected, the dataset X_t with m samples is projected into

the PCA space by performing the operation of $Y = X_t$*C where C is an *n*-by-Ƈ matrix containing the Ƈ eigenvectors. *Y* then is an *m*-by-Ƈ matrix representing the complete training set mapped into the PCA space, where each row contains the projected training samples. The inverted covariance matrix of *Y* is computed, yielding a highly reduced Ƈ-by-Ƈinverse covariance matrix Σ'^{-1}, as compared to the larger *n*-by-*n* Σ^{-1}. In addition, a 1-by-Ƈ mean vector $\mu'$is computed from the training samples Y. Note that all variables with the $'$ symbol are mapped into PCA space. These steps yield a more efficient anomaly detection model $\mathcal{N}(\mu', \Sigma')$ in the PCA space, with the computational complexity being $O(\text{Ƈ}^3) \ll O(n^3)$. Figure 38 shows the complexity of the PCA anomaly detector with respect to the number of principal components retained. For example, if the initial feature vector is of length n=200, the complexity would be 8e6. However, if only 50 principal components were needed for acceptable performance, the computational complexity is about 1e5, which is a significant reduction by a factor of 80.

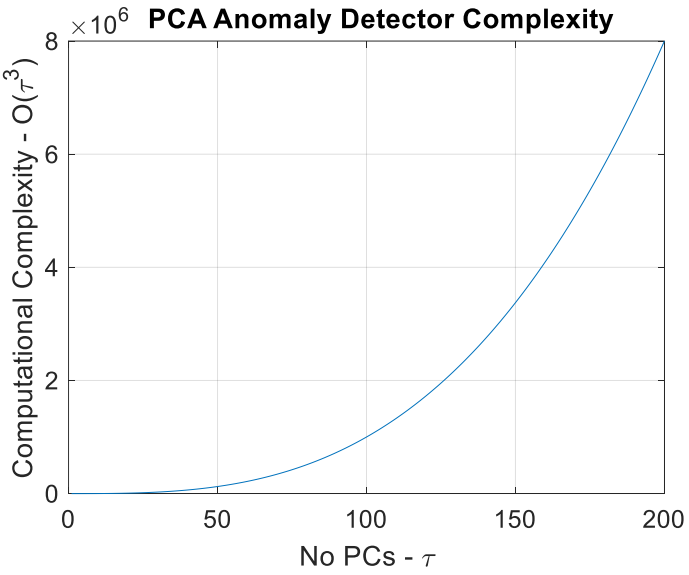


Figure 38. PCA Anomaly Detector - Complexity Verses Number of Principal Components

The anomaly detection algorithm for a given feature vector x from the test data is summarized in the following equations (6) through (9): by first performing a normal transformation

is required $x' = ntransform(x) * C,$ then computing the distance $D(x') = (x' - \mu')^T \Sigma'^{-1}(x' - \mu')$, and then applying the hypothesis tests: $H_o' : dmin' \leq D(x') \leq dmax',\ x' \epsilon \mathcal{N}(\mu', \Sigma')$ to declare a known class, or $H_1' : D(x') < dmin' or\ D(x') > dmax,\ x' \notin \mathcal{N}(\mu', \Sigma')$ to declare an anomaly.

For training the anomaly detector, the hyperparameters are the number of principal components C; minimum distance $dmin'$; and maximum distance $dmax'$. As starting points, $dmin'$ and $dmax'$ can be chosen so that $dmin' \cong \mu_{D(Y)} - A\sigma_{D(Y)}$ and $dmax' \cong \mu_{D(Y)} + A\sigma_{D(Y)}$. The scalar values $\mu_{D(Y)}$ and $\sigma_{D(Y)}$, are the mean and standard deviation of the 1-by-m Mahalanobis distance vector $D(Y)$. Finally, the scalar A is a hyper-parameter used to define the decision boundary with respect to $\mu_{D(Y)}$ and $\sigma_{D(Y)}$.

To carry out the experiment, PCA anomaly detector models were trained for each CNN according to the previous discussions by using all nine of the known classes of the training data sets, with SNR levels ranging from -6 to 14 dB. Assuming a normal distribution, the scalar A was set to four (4) to guarantee 99.9 percent of the known data points are correctly identified as knowns. Fixing A to a constant value for all models is not likely optimal, but the goal is to simplify this experiment to demonstrate reasonable performance and not to spend excessive time optimizing A for each model.

4.4 DATASETS

Three waveform datasets were used to carry out this experiment. The first two datasets were the known training and test waveform data. The training data was used to train the PCA anomaly detector models, and the test data was used for testing performance on recognizing known

waveforms. The third dataset is the unknown or anomaly waveforms used to assess the performance of the PCA anomaly detectors. The following subsections discuss these datasets in further detail.

4.4.1 PCA Training & Test Dataset

The training dataset the PCA anomaly detector is composed of nine (9) distinct known waveform classes with SNR levels ranging from -6 to 14 dB. There were 750 training samples for each of the nine-known classes, and 300 test samples for each of the nine-known classes. The PCA anomaly detection models were trained using the known training dataset, and the test dataset was used for testing performance in recognizing known waveforms. The known test data set contains 9*300=2700 total test samples. The complete details of the training and test datasets are as described in section 3.4.

4.4.2 Anomaly Dataset

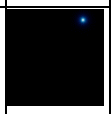
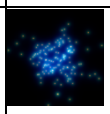
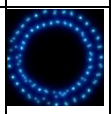
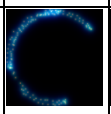
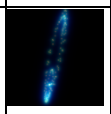
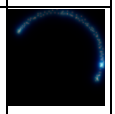
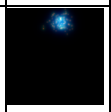
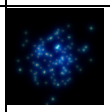
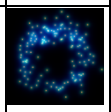
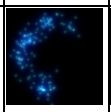
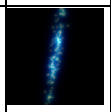
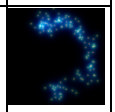
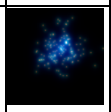
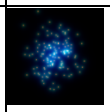
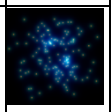
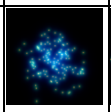
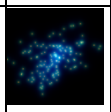
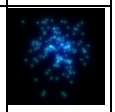
The anomaly dataset is used to test the performance of the PCA anomaly detectors. The six (6) modulation types were M-DSB, AM-SSB, CPFSK, GFSK, and PAM4. There are one thousand (1000) symbols per sample. With a dB step size of 2 going from -6 to 14 dB, this amounts to eleven (11) SNR levels ranging from -6, -4, … 14 dB.

Table 11. Dataset Summary (Anomalies)

Modulation Type	Total Samples	Symbols per Sample	SNR Min	SNR Max	SNR Step	#SNR Levels	Samples Per SNR
AM_DSB	550	1000	-6	14	2	11	50
AM_SSB	550	1000	-6	14	2	11	50
CPFSK	550	1000	-6	14	2	11	50
GFSK	550	1000	-6	14	2	11	50
PAM4	550	1000	-6	14	2	11	50
WBFM	550	1000	-6	14	2	11	50

The data was extracted from the Deep Signal RadioML 2016.10A [24] dataset by randomly selecting sample subsets. Table 12 provides 3-Channel image samples of the six waveform types used as anomalies to measure performance of the PCA anomaly detectors.

Table 12. Anomaly Samples

SNR (dB)	AM_DSB	AM_SSB	CPFSK	GFSK	PAM4	WBFM
14						
4						
-6						

4.5 EVALUATION

The detailed performance results of the five (5) CNN architectures using the PCA based anomaly detector are presented in sections 4.5.1 through 4.5.5 with each section containing two subsections.

The first subsection analyzes with respect to the number of principal components retained, with a series of anomaly detector probability distributions generated from the training, test, and anomaly data. Each plot displays the mean and standard deviation of the Mahalanobis distance of the data sets in the PCA space, and these parameters are used to define the lower and upper decision boundary thresholds shown at the top of the dashed vertical lines in each plot (ant negative decision

boundary values are set to zero because distance calculations are always positive). The distributions are useful for visualizing the PCA model trends with respect to the number of principal components used.

The second subsection provides summary tables and plots of the performance of each anomaly detector, with results reported with respect to the number of positives (p) defined as the number of known test samples, and the number of negatives (n) defined as the number of anomaly samples. The definitions below summarize the meaning of the tables' column heading labels.

- (n) is an integer defining the number of anomaly samples
- (p) is an integer defining the number of known samples
- True positive (TP) is the number of correctly identified knowns.
- True negatives (TN) are the number of correctly identified anomalies (unknowns).
- False positives (FP) are the number of incorrectly identified knowns.
- False negatives (FN) are the number of incorrectly identified anomalies (unknowns).
- The true positive rate (TPR) is defined as TP/p.
- The true negative rate (TNR) is defined as TN/n.
- The false negative rate (FNR) is defined as FN/n.
- The overall accuracy (ACC) is defined as (TP+TN)/(p+n).
- Positive prediction value (PPV) is defined as TP/(n+FN)
- The number of Principal components retained for the model (NoPCs)
- Percentage of PCA variance (PVAR), variance contribution resulting from the number of principal components retained (NoPCs).

Section 4.5.6 provides an overall summary of the results, and experimentation showed that applying normal transformations were not necessary to obtain reasonable performance, therefore the feature vector outputs at the CNN classification layers are assumed normally distributed.

4.5.1 AlexNet

A series of PCA based discrete probability density plots of the training, test, and anomaly data at the classification layer of AlexNet follow. Each plot is with respect to the number of principal components retained for the model. One can see in **Error! Reference source not found.** with PCs = 2, that the datasets are inseparable regardless of the selected threshold boundaries because the distances are in the same range as the known and test values. This trend continues for the number of PCs = 4 and 8 as shown in **Error! Reference source not found.** and Figure 41. For the remaining figures, Figure 42 through Figure 49, separation becomes more evident as the number of PCs increases. One can also see that as the number of PCs increases, using a fixed decision boundary of 4 sigma provides reasonable performance, however it is not optimal. There are cases where the decision boundaries could have been widened to increase performance. This is apparent in the figures where the number of PCs ranges from 256 to 2048.

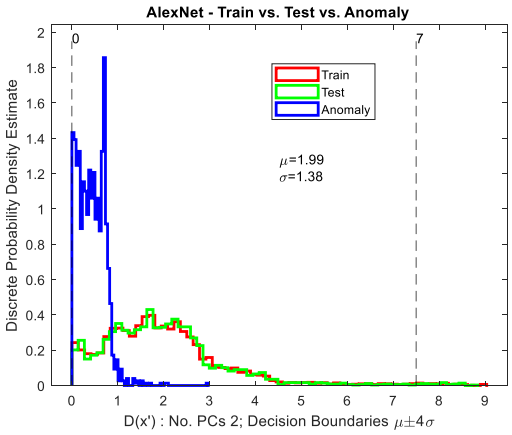


Figure 39. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs =2

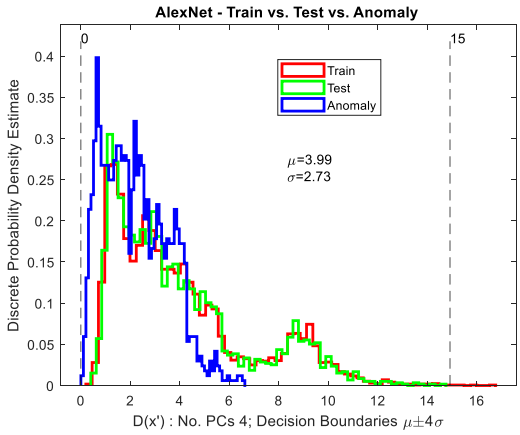


Figure 40. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 4

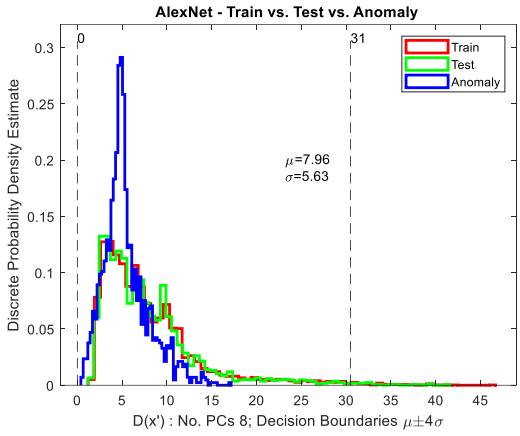


Figure 41. AlexNet- PCA Mahalanobis Discrete Probability Distributions, PCs = 8

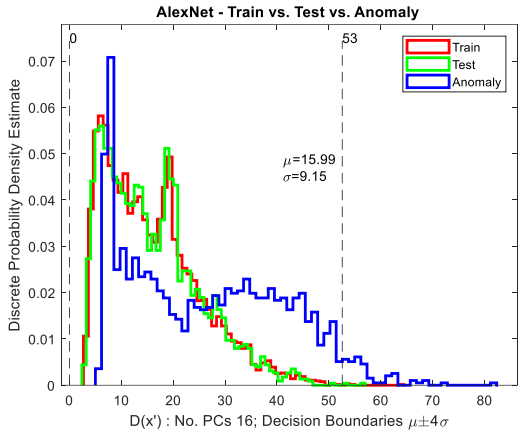


Figure 42. AlexNet - PCA Mahalanobis Discrete Probability Distribution, PCs = 16

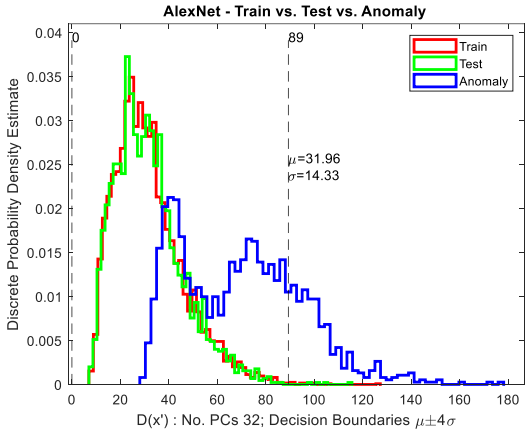


Figure 43. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 32

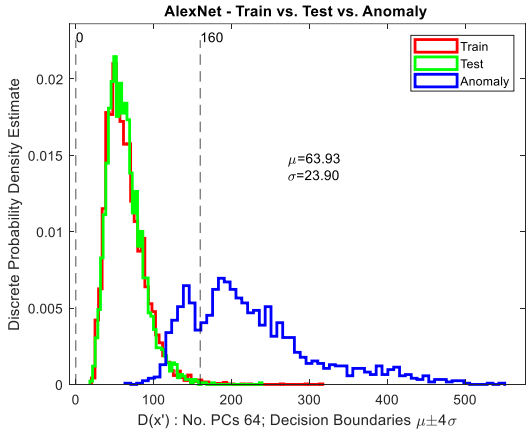


Figure 44. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 64

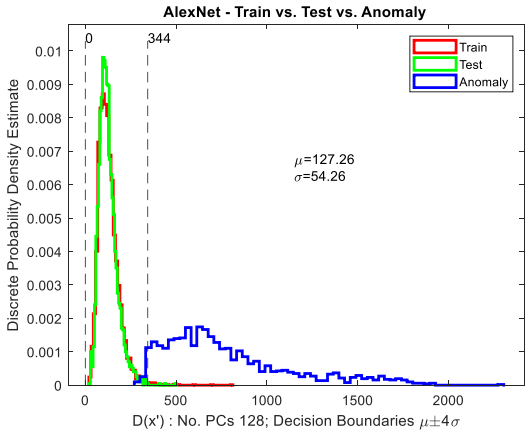


Figure 45. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 128

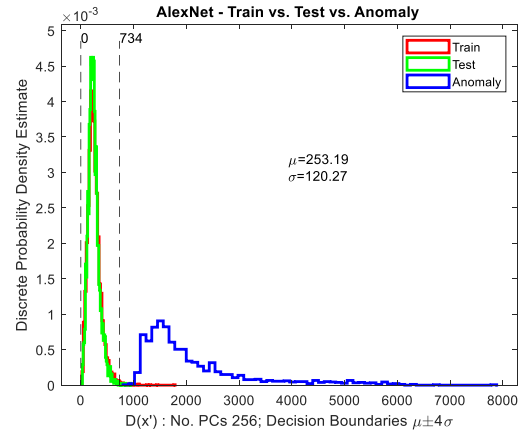


Figure 46. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 256

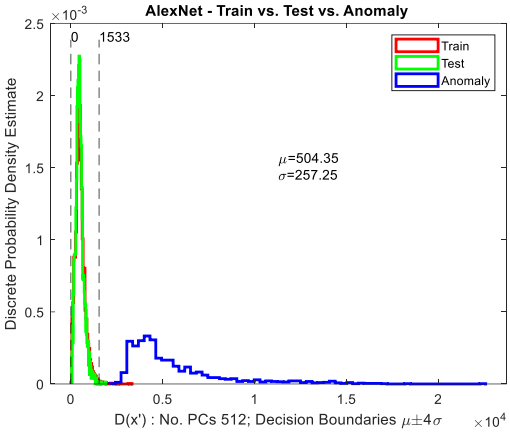


Figure 47. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 512

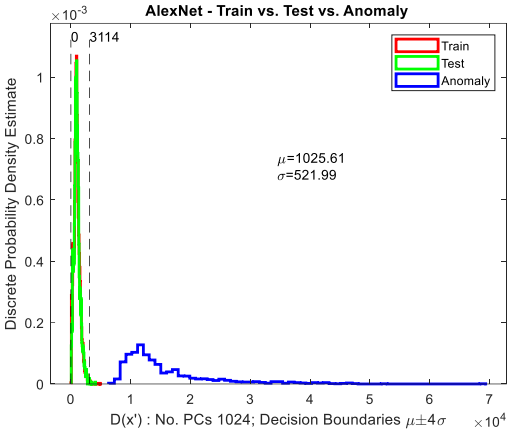


Figure 48. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 1024

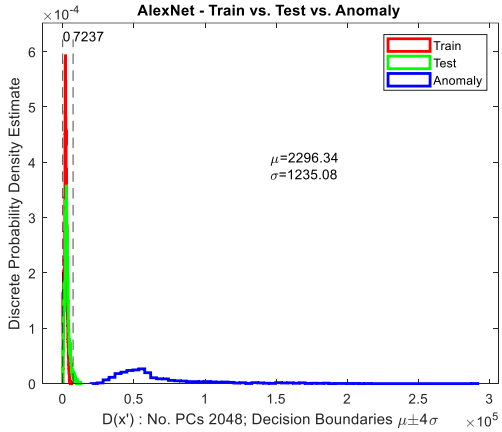


Figure 49. AlexNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 2048

Table 13 shows the performance results of the AlexNet PCA anomaly detector with respect to the selected number of principal components. This table is related to the probability distribution figures shown in the previous section. Acceptable performance is obtained when using 128 principal components or where the TPR and TNR values are greater than 90%. Figure 50 shows a

plot of the data in Table 13. The reduction of accuracy as the PCs reach 1024 and 2048 can be attributed to non-optimal thresholds.

Table 13. AlexNet - PCA Anomaly Detection Performance

NoPcs	ACC	TPR	TNR	FPR	FNR	PVAR
2	61.40	98.93	0.00	100.00	1.07	48.87
4	62.07	100.00	0.00	100.00	0.00	63.65
8	61.59	99.22	0.00	100.00	0.78	74.98
16	63.47	99.93	3.82	96.18	0.07	84.78
32	70.60	99.85	22.73	77.27	0.15	90.72
64	91.29	99.85	77.27	22.73	0.15	94.05
128	99.03	99.85	97.70	2.30	0.15	96.19
256	99.91	99.85	100.00	0.00	0.15	97.68
512	99.93	99.89	100.00	0.00	0.11	98.77
1024	99.91	99.85	100.00	0.00	0.15	99.56
2048	98.30	97.26	100.00	0.00	2.74	99.96

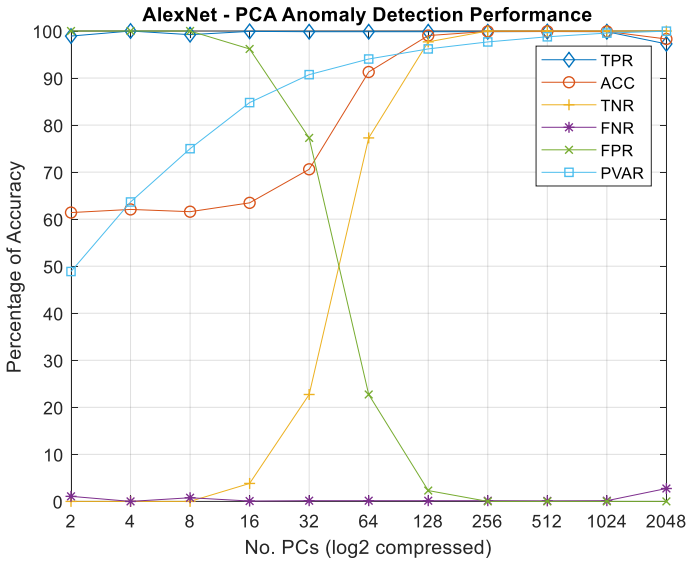


Figure 50. AlexNet - PCA Anomaly Detection Performance

4.5.2 Inception V3

A series of PCA discrete probability density plots of the training, test, and anomaly data at the classification layer of Inception V3 follow. Each plot is with respect to the number of principal components retained for the model. In Figure 51 and Figure 52 with PCs = 2 & 4, the datasets are

inseparable. Figure 53 through Figure 55, show the minimal separation evolving as the number of PCs increases from 8 to 32 PCs. However, in Figure 56 and Figure 57 the datasets become inseparable as the number of PCs is set to 64 and 128. This indicates that the 128 chosen PCs are not sufficient to represent the relevant features that distinguish normal from anomalous data. In the remaining figures Figure 58 Figure 58 through Figure 61, with PCs set from 256 to 2028, separability becomes increasingly apparent.

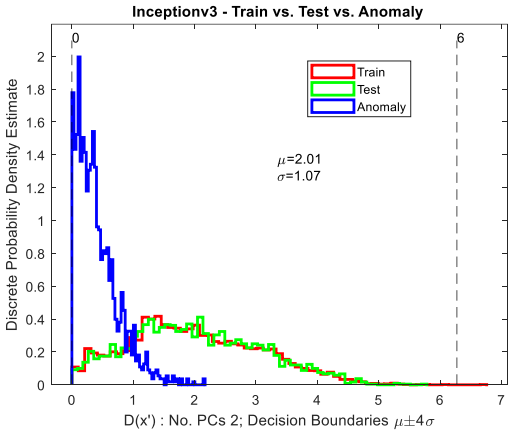


Figure 51. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 2

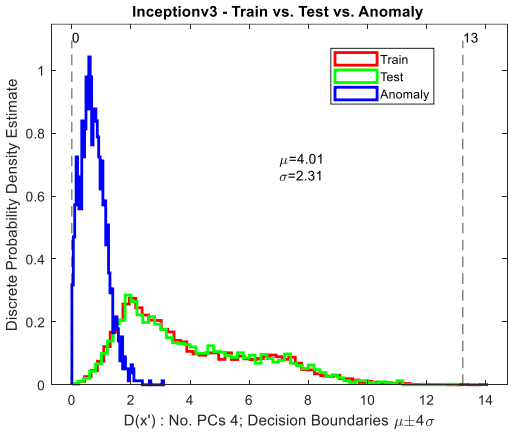


Figure 52. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 4

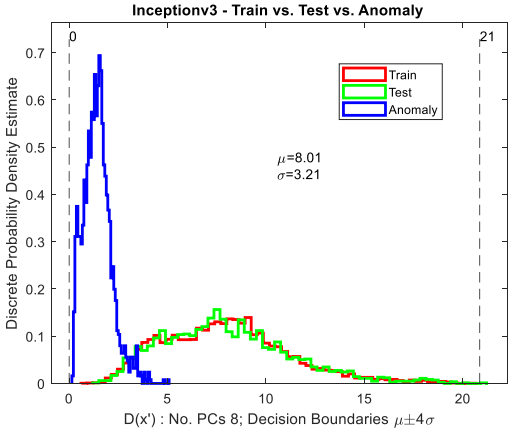


Figure 53. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 8

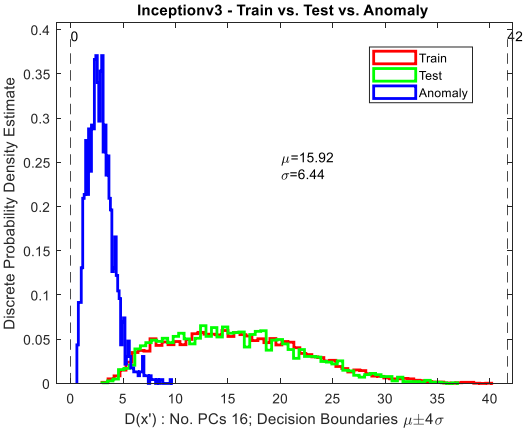


Figure 54. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 16

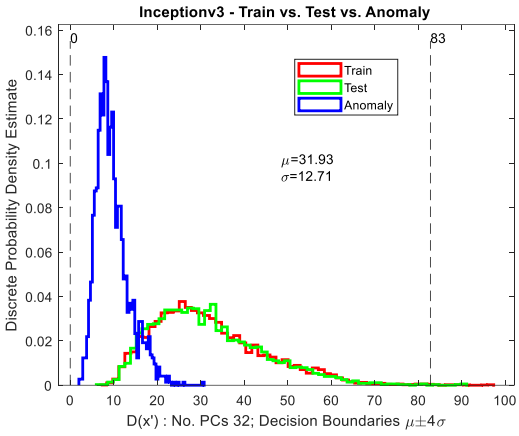


Figure 55. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 32

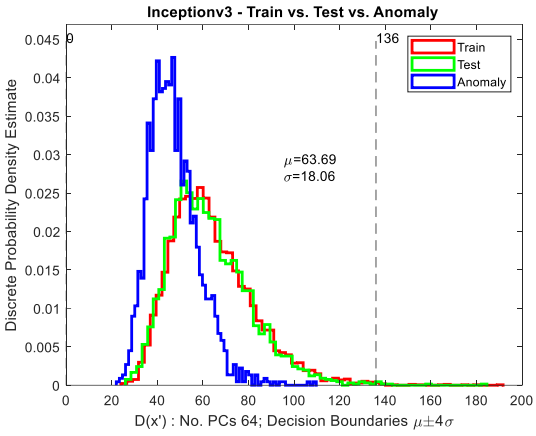


Figure 56. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 64

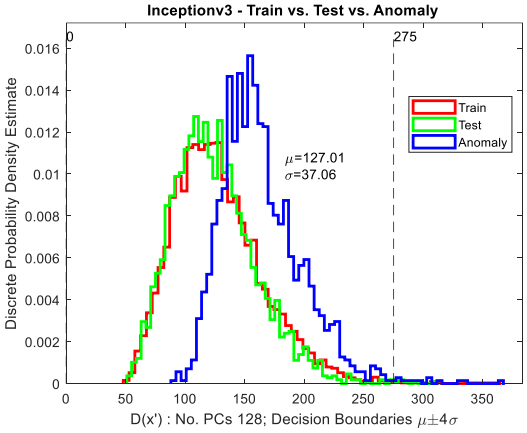


Figure 57. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 128

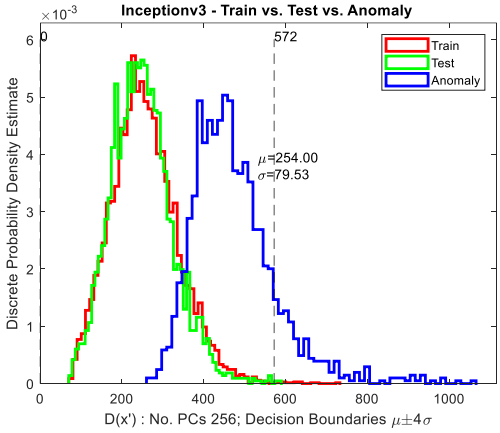


Figure 58. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 256

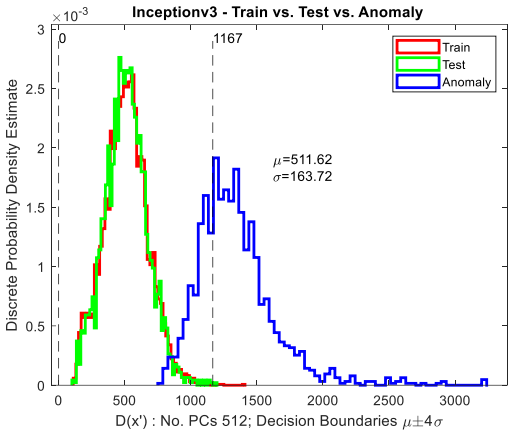


Figure 59. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 512

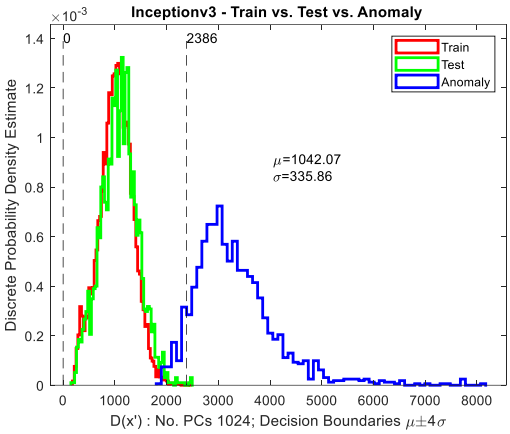


Figure 60. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 1024

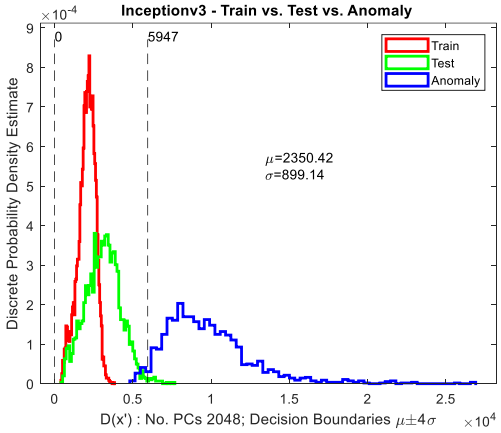


Figure 61. Inception - PCA Mahalanobis Discrete Probability Distributions, PCs = 2048

Table 14 shows the performance results of the Inception V3 PCA based anomaly detector with respect to the selected number of principal components. Acceptable performance is obtained when using 1024 principal components where the TPR and TNR values are greater than 90%. Figure 62 shows a plot of the data in Table 14.

Table 14. Inception - PCA Anomaly Detection Performance

NoPcs	ACC	TPR	TNR	FPR	FNR	PVAR
2	62.07	100.00	0.00	100.00	0.00	16.28
4	62.07	100.00	0.00	100.00	0.00	27.84
8	62.05	99.96	0.00	100.00	0.04	43.61
16	62.07	100.00	0.00	100.00	0.00	60.52
32	61.98	99.85	0.00	100.00	0.15	71.54
64	61.91	99.74	0.00	100.00	0.26	78.64
128	62.23	99.78	0.79	99.21	0.22	85.03
256	67.15	99.93	13.52	86.48	0.07	91.01
512	89.40	99.96	72.12	27.88	0.04	95.82
1024	97.20	99.85	92.85	7.15	0.15	98.74
2048	98.30	99.00	97.15	2.85	1.00	100.00

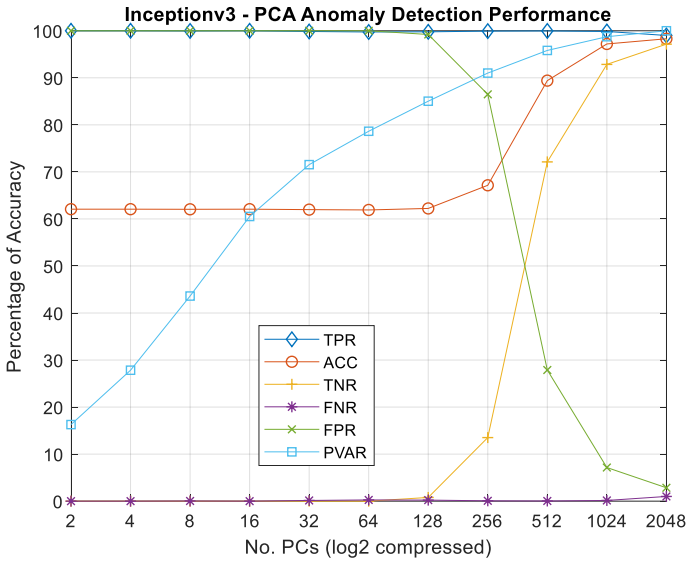


Figure 62. Inception - PCA Anomaly Detection Performance

4.5.3 GoogleNet

A series of PCA discrete probability density plots of the training, test, and anomaly data at the classification layer of GoogleNet follows. Each plot is with respect to the number of principal components retained for the model. In Figure 63 through Figure 66, with PCs = 2, 4, 8, and 16, the datasets are inseparable. In Figure 67 through Figure 72 the datasets become increasingly separable as the number of PCs increases from 32 through 2048.

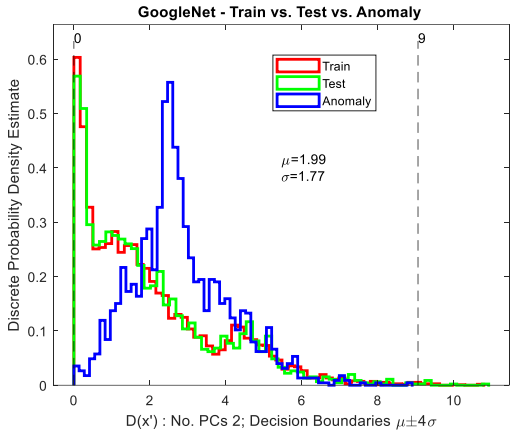


Figure 63. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 2

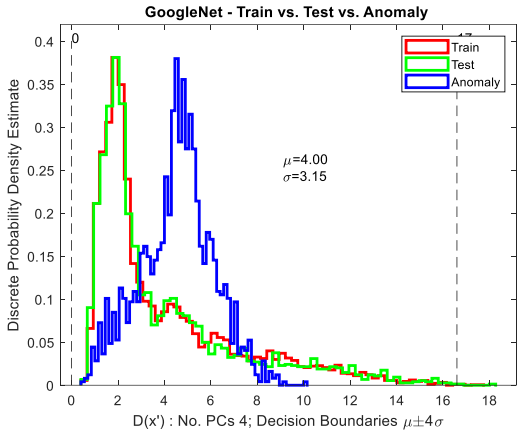


Figure 64. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 4

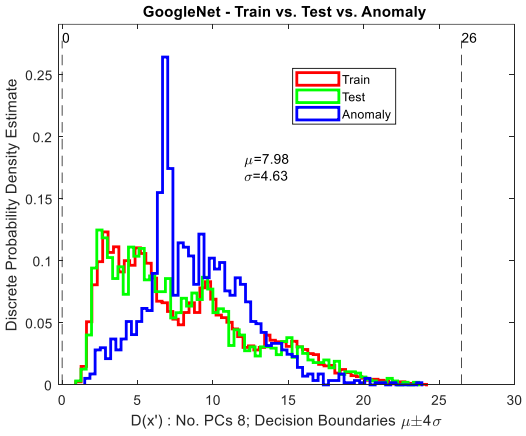


Figure 65. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 8

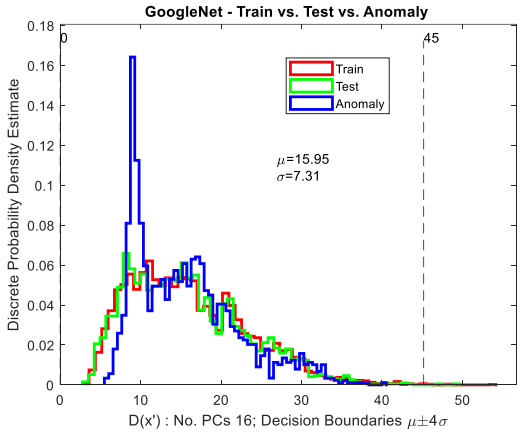


Figure 66. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 16

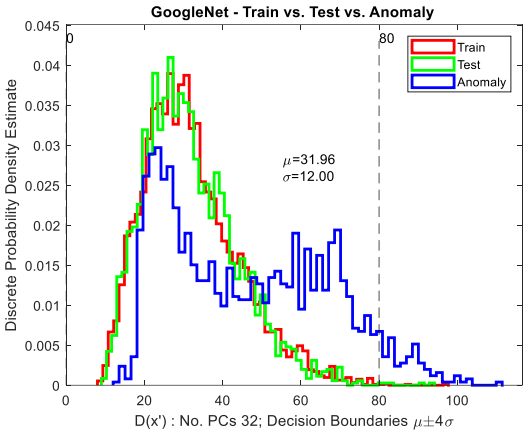


Figure 67. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 32

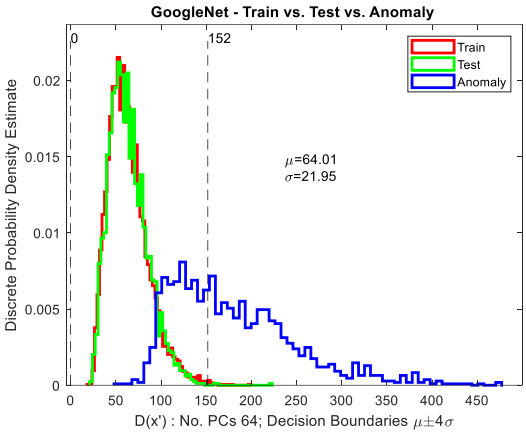


Figure 68. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 64

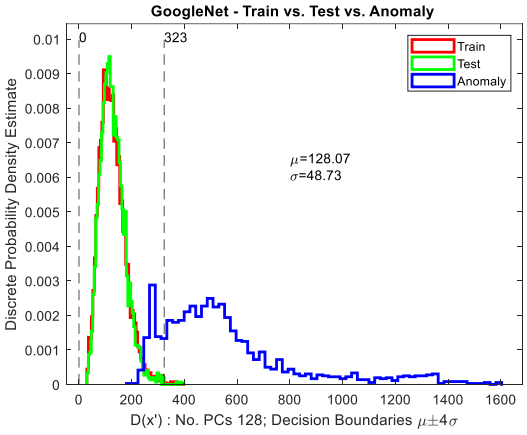


Figure 69. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 128

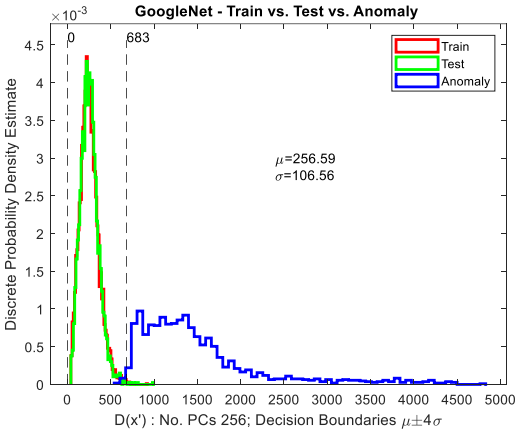


Figure 70, GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 256

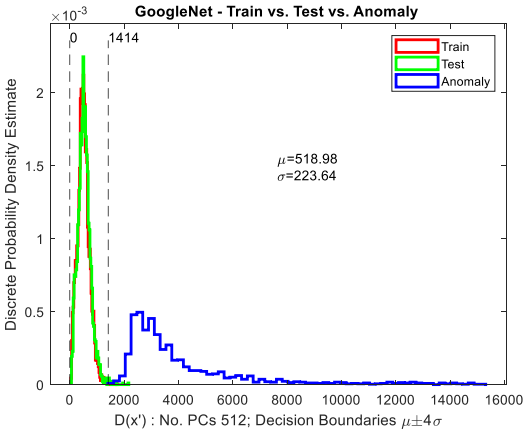


Figure 71. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 512

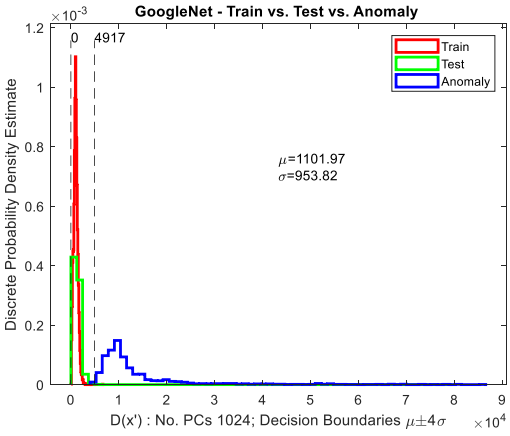


Figure 72. GoogleNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 1024

Note how as the number of pcs increases the training and test data are confined to a tighter distribution whereas the anomalous data has a much wider spread. Table 15 shows the performance results of the GoogleNet PCA base anomaly detector with respect to the selected number of principal components. Overall acceptable performance is obtained when using 256 principal components or more where the TPR and TNR values are greater than 90%. Figure 73 shows a plot of the data in Table 15.

Table 15. GoogleNet - PCA Anomaly Detection Performance

NoPcs	ACC	TPR	TNR	FPR	FNR	PVAR
2	61.91	99.74	0.00	100.00	0.26	32.87
4	62.02	99.93	0.00	100.00	0.07	52.08
8	62.07	100.00	0.00	100.00	0.00	71.12
16	62.02	99.93	0.00	100.00	0.07	84.64
32	64.23	99.81	6.00	94.00	0.19	91.49
64	84.07	99.89	58.18	41.82	0.11	95.15
128	94.28	99.89	85.09	14.91	0.11	97.43
256	99.54	99.74	99.21	0.79	0.26	98.87
512	99.70	99.52	100.00	0.00	0.48	99.69
1024	99.77	99.89	99.58	0.42	0.11	100.00

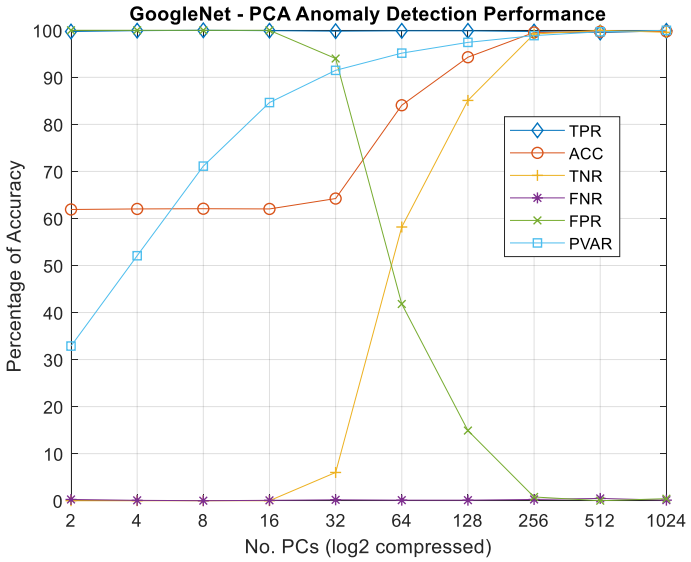


Figure 73. GoogleNet - PCA Anomaly Detection Performance

4.5.4 ResNet50

A series of PCA discrete probability density plots of the training, test, and anomaly data at the classification layer of ResNet50 follow in this section. Each plot is with respect to the number of principal components retained for the model. In Figure 74 through Figure 78, with PCs = 2, 4, 8, 16, and 32, the datasets are generally inseparable. In Figure 79 through Figure 84 the datasets become increasingly separable as the number of PCs increases from 64 through 2048.

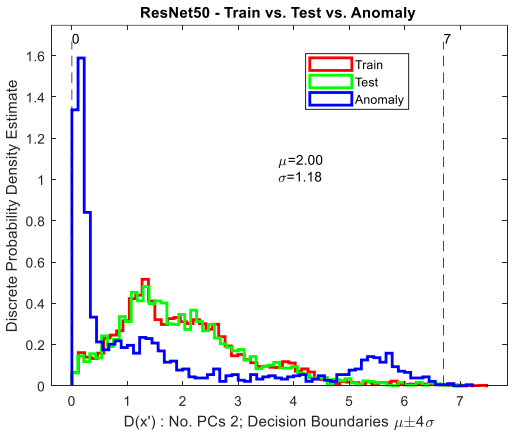


Figure 74. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 2

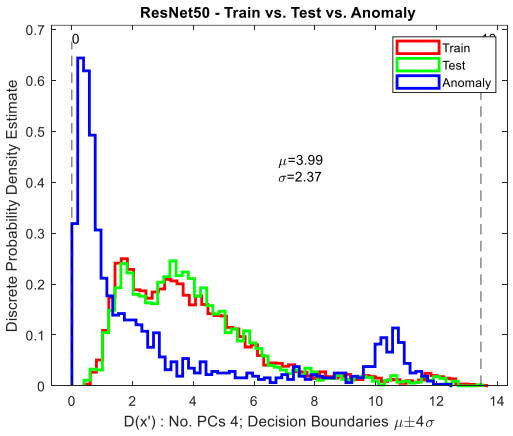


Figure 75. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 4

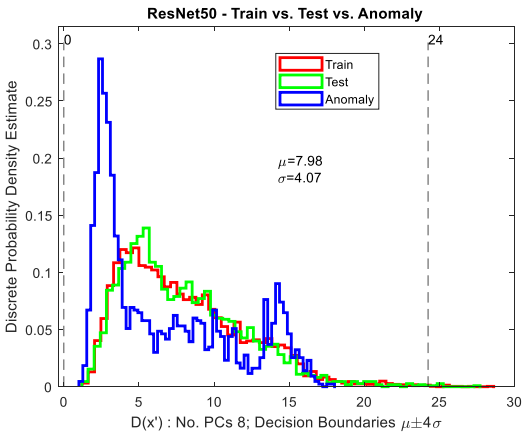


Figure 76. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 8

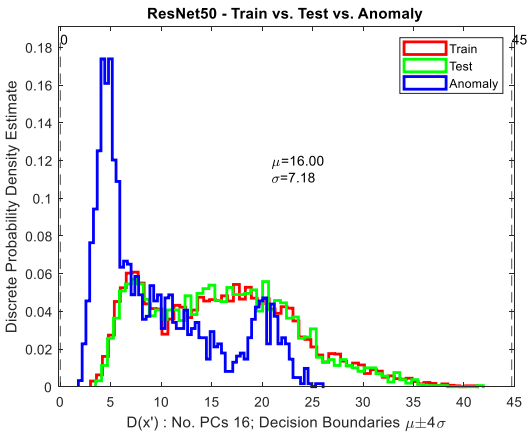


Figure 77. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 16

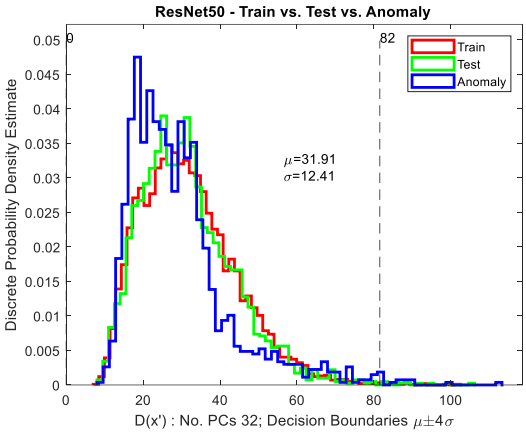


Figure 78. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 32

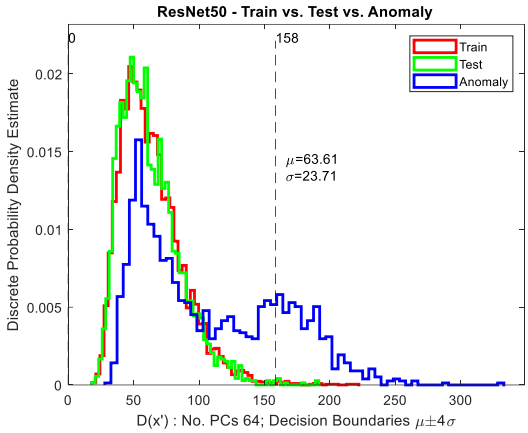


Figure 79. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 64

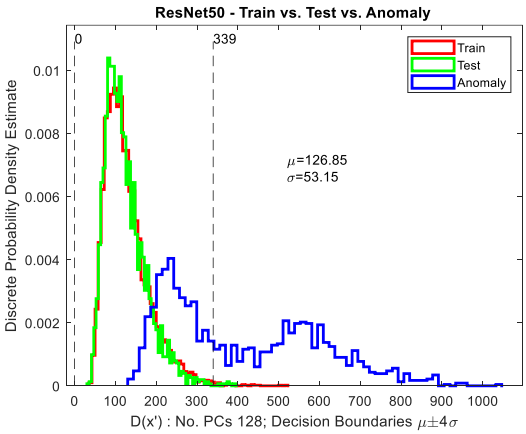


Figure 80. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 128

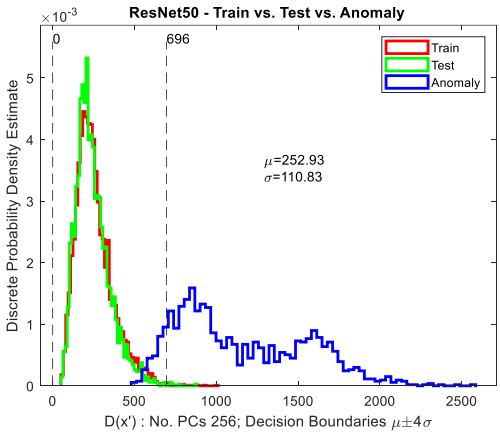


Figure 81. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 256

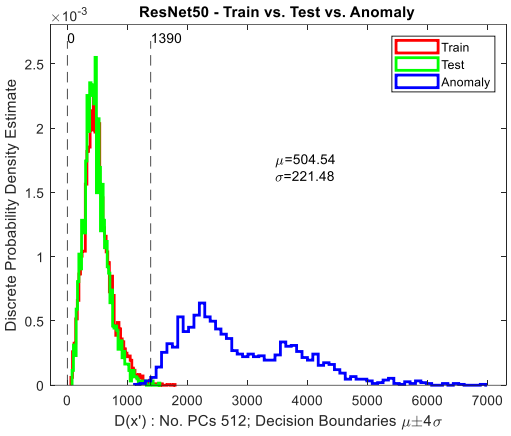


Figure 82. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 512

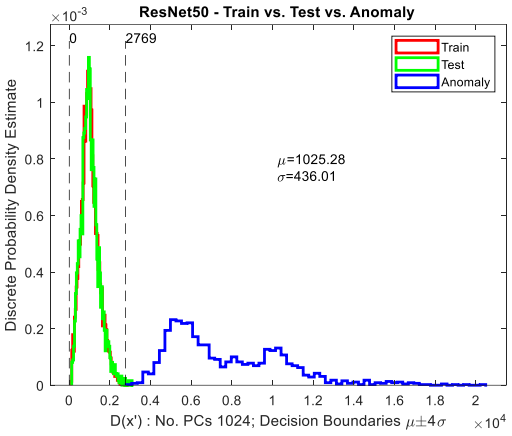


Figure 83. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 1024

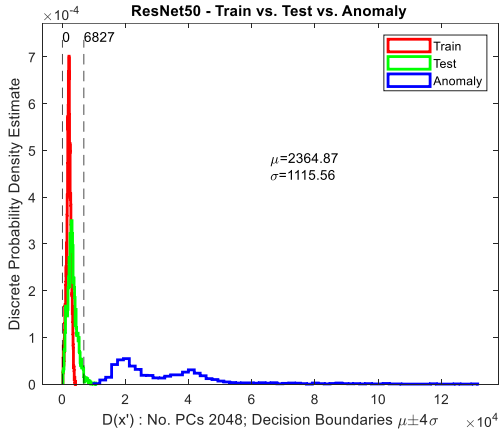


Figure 84. ResNet50 - PCA Mahalanobis Discrete Probability Distributions, PCs = 2048

Table 16 shows the performance results of the ResNet 50 PCA base anomaly detector with respect to the selected number of principal components. Acceptable performance is obtained when using 256 principal components and above where the TPR and TNR values are greater than 90%. Again, as the number of PCs is increased beyond 1024, performance drops as seen in ***Table 16***, due to non-optimal thresholds. Figure 85 shows a plot of the data in Table 16.

Table 16. ResNet 50 - PCA Anomaly Detection Performance

NoPcs	ACC	TPR	TNR	FPR	FNR	PVAR
2	62.11	99.93	0.24	99.76	0.07	23.44
4	62.07	100.00	0.00	100.00	0.00	37.12
8	61.93	99.78	0.00	100.00	0.22	54.02
16	62.07	100.00	0.00	100.00	0.00	70.86
32	62.14	99.67	0.73	99.27	0.33	80.52
64	70.80	99.74	23.45	76.55	0.26	86.21
128	82.05	99.74	53.09	46.91	0.26	90.62
256	96.46	99.81	90.97	9.03	0.19	94.15
512	99.72	99.85	99.52	0.48	0.15	96.88
1024	99.89	99.81	100.00	0.00	0.19	98.90
2048	98.76	98.00	100.00	0.00	2.00	100.00

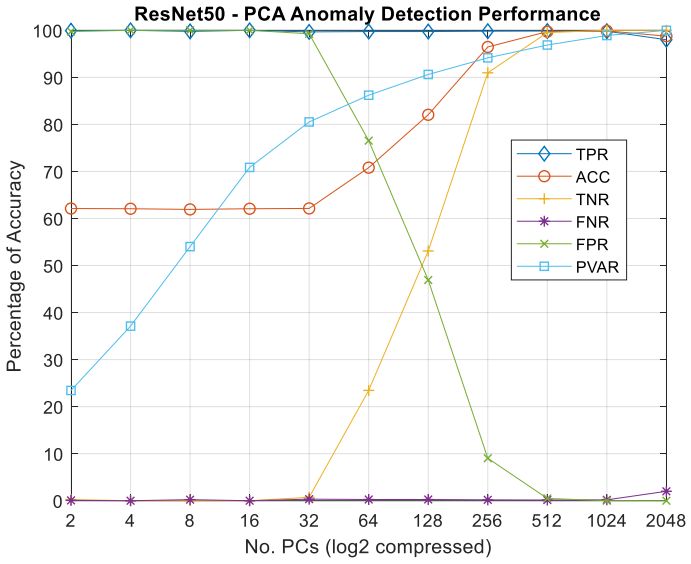


Figure 85. ResNet50 - PCA Anomaly Detection Performance

4.5.5 MobileNet

A series of PCA discrete probability density plots of the training, test, and anomaly data at the classification layer of MobileNet follow in this section. Each plot is with respect to the number of principal components retained for the model. In Figure 86 through Figure 91, with PCs = 2, 4, 8, 16, 32, and 64 the datasets are generally inseparable. In Figure 92 through Figure 95 the datasets are becoming increasingly separable as the number of PCs increases from 128 through 2048.

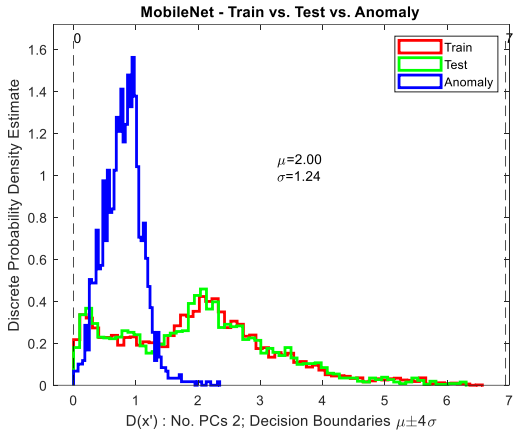


Figure 86. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 2

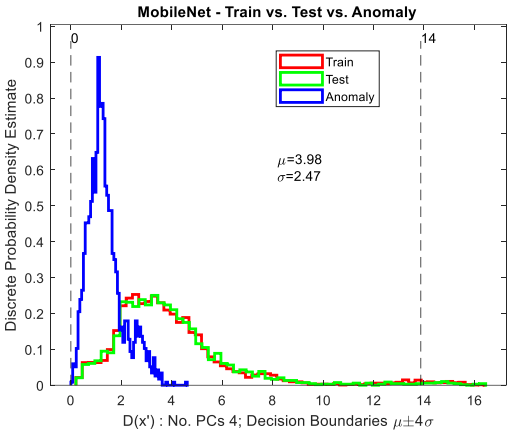


Figure 87. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 4

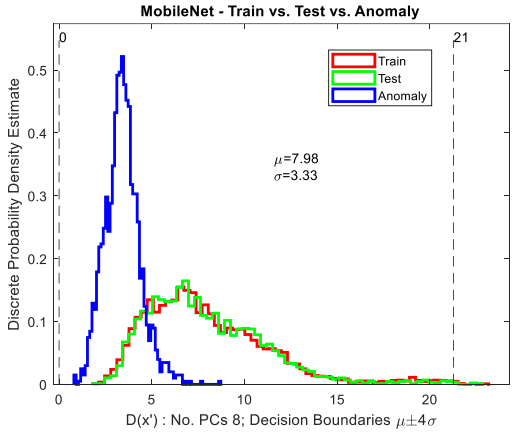


Figure 88. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 8

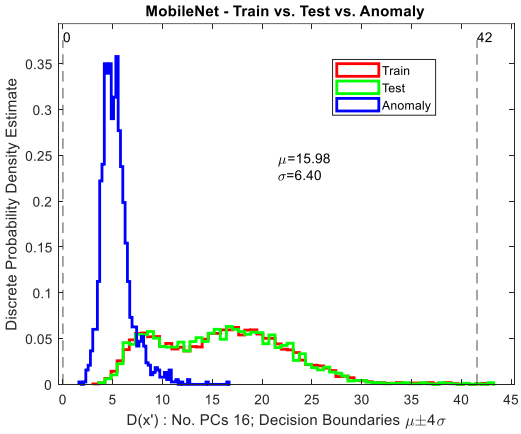


Figure 89. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 16

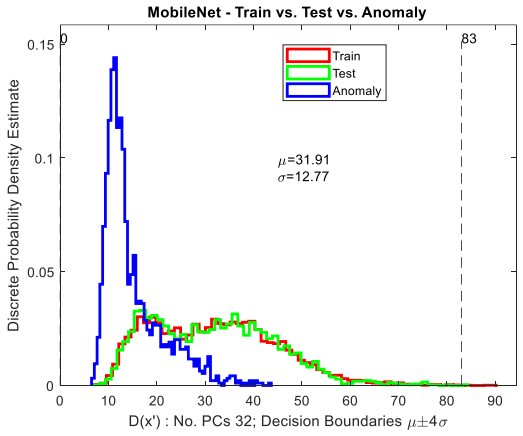


Figure 90. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 32

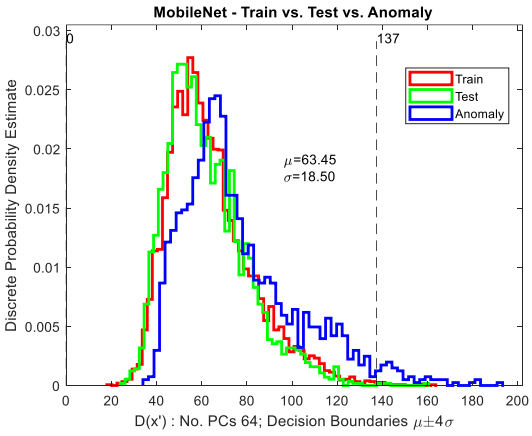


Figure 91. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 64

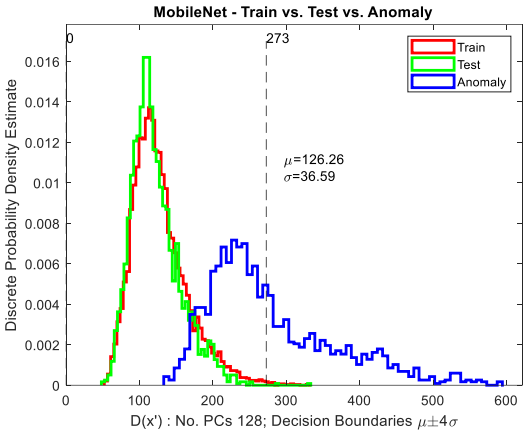


Figure 92, MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 128

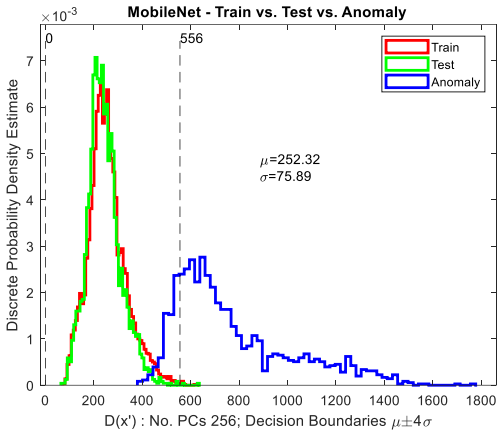


Figure 93, MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 256

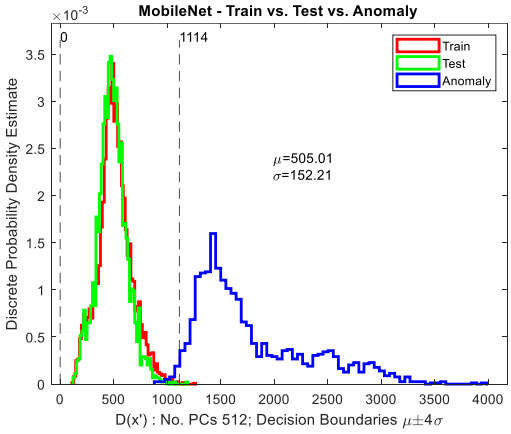


Figure 94. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 512

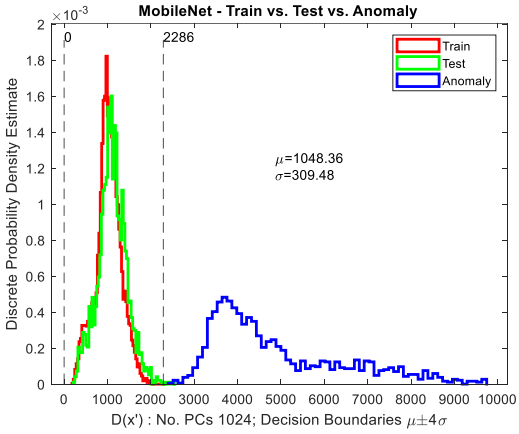


Figure 95. MobileNet - PCA Mahalanobis Discrete Probability Distributions, PCs = 1024

Table 17 shows the performance results of the MobileNet PCA based anomaly detector with respect to the selected number of principal components. Acceptable performance is obtained when using 512 principal components and above where the TPR and TNR values are greater than 90%. Figure 96 shows a plot of the data in Table 17.

Table 17. MobileNet - PCA Anomaly Detection Performance

NoPcs	ACC	TPR	TNR	FPR	FNR	PVAR
2	61.88	100.00	0.00	100.00	0.00	18.92
4	61.22	98.93	0.00	100.00	1.07	31.70
8	61.80	99.88	0.00	100.00	0.12	48.96
16	61.80	99.88	0.00	100.00	0.12	64.28
32	61.86	99.96	0.00	100.00	0.04	74.25
64	63.13	99.88	3.47	96.53	0.12	80.57
128	76.87	99.92	39.47	60.53	0.08	86.21
256	93.24	99.92	82.40	17.60	0.08	91.47
512	99.26	99.96	98.13	1.87	0.04	95.85
1024	99.90	99.84	100.00	0.00	0.16	99.28

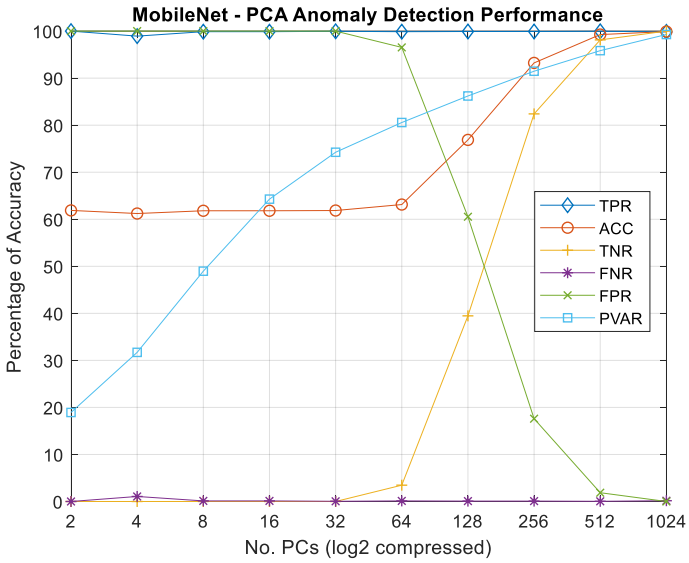


Figure 96. MobileNet - PCA Anomaly Detection Performance

4.5.6 Overall Accuracy Aggregation

Table 18 shows an aggregation of the overall accuracy (ACC) results for each of the CNNs' PCA anomaly detectors, with a dash entry indicating that the CNN does not have a feature vector of NoPcs size.

Figure 97 shows a plot of the Table 18 results.

Table 18. CNN PCA Accuracy Aggregation

NoPcs	AlexNet	InceptionV3	GoogleNet	ResNet 50	MobileNet
2	61.40	62.07	61.91	62.11	61.88
4	62.07	62.07	62.02	62.07	61.22
8	61.59	62.05	62.07	61.93	61.80
16	63.47	62.07	62.02	62.07	61.80
32	70.60	61.98	64.23	62.14	61.86
64	91.29	61.91	84.07	70.80	63.13
128	99.03	62.23	94.28	82.05	76.87
256	99.91	67.15	99.54	96.46	93.24
512	99.93	89.40	99.70	99.72	99.26
1024	99.91	97.20	99.77	99.89	99.90
2048	98.30	98.30	-	98.76	-

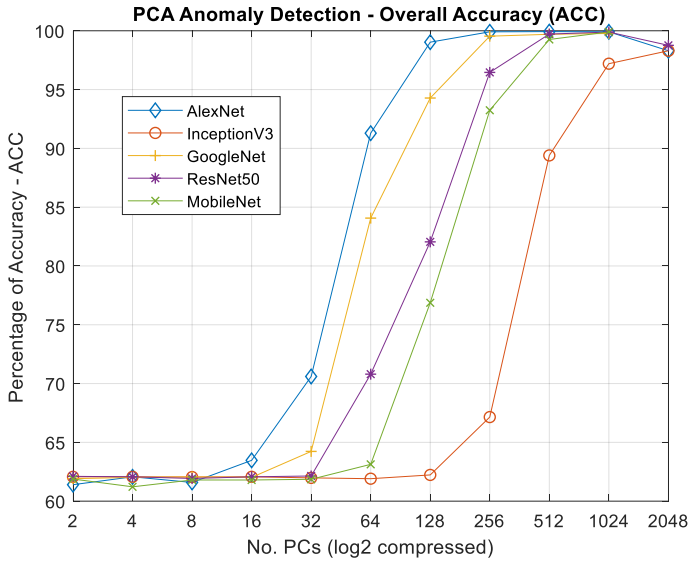


Figure 97. PCA Anomaly Detection - Overall Accuracy

For all anomaly detectors it can be observed from the table and figure that in general, as the number of PCs increases, the overall accuracy tends to increase. However, there are the three cases, AlexNet, ResNet50, and Inception V3, where when the number of PCs equals 2048, the accuracy drops. This drop is surmised to be attributed to over-fitting of the PCA detectors on the training data as more PCA components are used. Listed in the order of the best overall accuracy of performance, AlexNet performs the greatest, followed by GoogleNet, ResNet50, MobileNet, and finally ResNet50. These findings are not intended to generalize that one network can outperform

another because that is not the scope of this study. In fact, because the experiment was simplified by selecting a fixed factor of 4 sigma for the decision boundaries, one can see from the discrete probability plots that there are various cases where using the factor of 4 is not optimal, and in those cases choosing a greater value would yield better performance in separating the anomalous data.

4.6 SUMMARY

This effort successfully demonstrated that accurate RF waveform anomaly detection is possible by monitoring transfer learned CNNs. PCA was successfully used at the last classification layer of the CNNs to detect anomalous waveforms, with highly accurate results. The PCA model was designed using the labeled training data. This effort demonstrated that a fixed classifier augmented with a PCA based anomaly detector can significantly improve overall performance in a non-stationary RF environment. This method also demonstrated that choosing a fixed 4 sigma boundary with the test data provides for reasonable but not necessarily optimal performance. This approach allows for preventing anomalous data from being classified as a known waveform therefore improving overall performance.

Although PCA anomaly detection has been shown can be highly accurate, the method has a shortfall in that the anomalous and the known datasets can actually map into the same PCA space if the number of principal components is not selected properly, therefore there is a computational and performance tradeoff when between choosing the number principal components, as increasing the number of principal components tends to increase accuracy but it comes at a cost of computational load.

Finally, a significant shortfall of this approach is that although it clearly can isolate anomalous data from labeled, it does not allow for online training, adaptation, and clustering of the anomalous data. The remaining work in this research will begin to address this problem.

5 RANDOM M-CLASS ANOMALY DETECTOR

This section describes work carried out using M randomly generated softmax or perceptron nodes monitoring the last layer of transfer learned CNNs with the goal of reliably detecting waveform anomalies at the last layer of CNNs for RF waveform data. This approach is motivated by ideas drawn from GNG architectures, where GNG has been widely cited in literature as very effective at anomaly detection[80]. In particular, the GNG network nodes are randomly initialized. This work is preliminary and explorative and has led to the ideas solidified in the adaptive classifier sections discussed later.

5.1 CHALLENGES

The challenge here is to carry out reliable anomaly detection on pretrained CNNs while maintaining the accuracy of performance on labeled data. A particular challenge of this approach that is not resolved in this section is a reliable initialization method for the M-class anomaly detector that allows a quick separation of labeled and unlabeled data.

5.2 CONTRIBUTIONS

This work has contributed to an easy to understand technique that manually adapts a classifier trained on known classes to detect and classify anomalous new classes. It serves as a baseline to compare against the automatic online adaptation techniques developed as part of this research and discussed in later sections.

5.3 METHODOLOGY

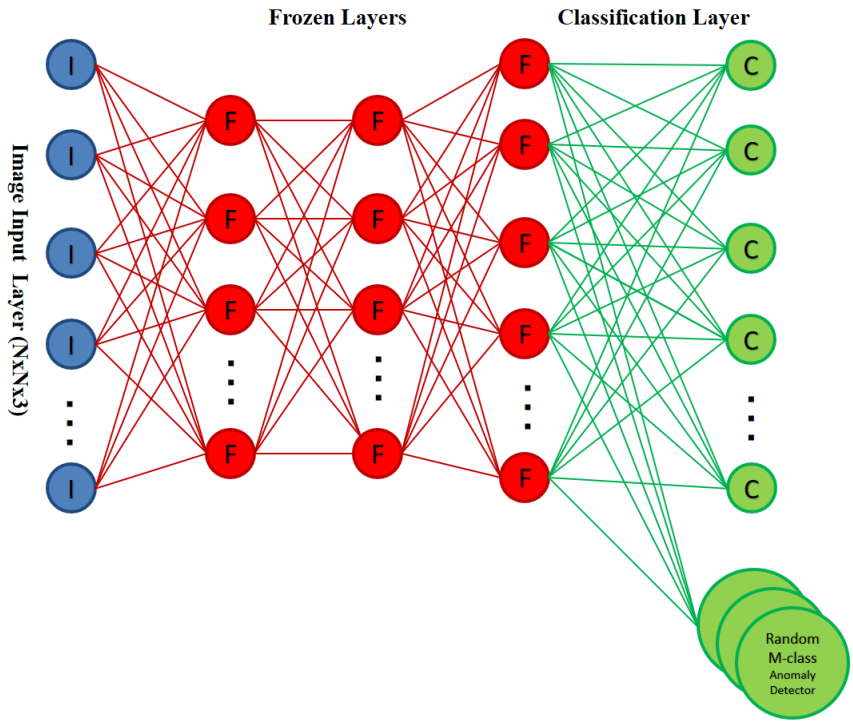


Figure 98. CNN Augmented with Random M-class Anomaly Detector

The 9-class CNNs are trained as discussed in section 2.6.2.3. Following training, the CNNs are modified by arbitrarily appending M = 1000, 500, or 100 softmax nodes to the classification layer of each network as depicted in Figure 98. The hypothesis for this approach having a reasonable chance of working is that when known signals are input to the network, they are highly probable of being selected by their associated trained class of weights and biases; however, when an unknown class is input to network, statistically it is more likely to be selected by the randomly assigned neurons since the unknown class feature vector is significantly different from the 9-class classifiers than the output layer was optimized upon. This is a fair assumption given that the anomalous WBFM waveform dataset images are significantly different from the known waveform datasets.

Testing variations on the number of nodes M is carried out to observe performance as a function of the number of random nodes utilized. The weights and biases of the M-class nodes are randomly assigned using normal distributions like the trained 9-class weights and biases. Experimentation demonstrated that the random weights and biases require similar distributions (normal distributions) to the 9-class classifier layer to perform reasonably well. Table 19 shows pseudo code for the procedure used to generate the M-class anomaly detection nodes. Further details of this work are also published in Conn et el. [101].

Table 19. Node Generation for Random M-Class Anomaly Detectors

```
1.  Initialize M = {1000, 500, 100}
2.  Foreach M = {1000, 500, 100}
3.    Foreach CNN = {AlexNet, GoogleNet, InceptionV3, ResNet50}
4.      net = load(CNN)
5.      Len = net.{length of fc classification layer};
6.      CurrentWeights={net.Layers(end-2).Weights}
7.
8.      //create M normally distributed biases
9.      BiasMean = mean(net.{last fc classification layer}.Bias);
10.     BiasStd= std(net.{last fc classification layer}.Bias);
11.     BiasRandom = StdBiasScale*normrnd(BiasMean,BiasStd,M);
12.
13.     //create M normally distributed weights
14.     WeightsMean = mean(net.Layers(end-2).Weights);
15.     WeightsStd= StdWeightScale*std(net.Layers(end-2).Weights);
16.     do
17.         WeightsRandom = normrnd(WeightsMean,WeightsStd, Len);
18.     While WeightsRandom in CurrentWeights
19.     WeightsRandom.Add(CurrentWeights)
20.
21.     //append weights & biases  to CNN structure
22.     CNNnew=Append(CNN, WeightsRandom(i,:), BiadRandom)
23.     Results=RunExperiment(CNNnew,testdata, anomalydata);
24.   endCNN
25. endM
```

5.4 DATASETS

To run this experiment two data sets were used. The first dataset consisted of nine known waveforms classes, and the second data set consisted of samples of a single anomaly waveform

class. A single anomaly class was used in this preliminary study to keep the experiment simple. The utilized datasets are described in more detail in the following two subsections.

5.4.1 Known Datasets

The known datasets used are as described in detail in section 3.4. For evaluations, the test data with high SNR levels ranging from 4 to 14 dB were used. High SNR levels were chosen to reduce noise effects, and to focus the experiment on isolating signals with high SNR levels. This in effect simplified analysis of the experiment.

5.4.2 Anomaly Dataset

Given that this experiment was a simple proof of concept, only one anomaly waveform class was used at a fixed 12 dB SNR level. To keep things simple, high SNR levels were chosen to reduce the noise effects. The WBFM waveforms were used because their constellation images were clearly distinct from the known waveform images. Figure 99 shows 6 samples of the WBFM anomaly waveform. Further details of this dataset and how it was generated are as described in section 4.4.2.

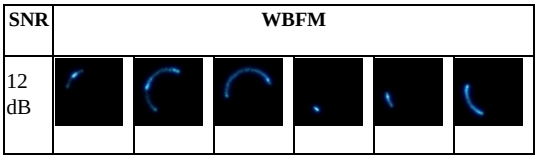


Figure 99. Anomaly Samples of 3-Channel Images - WBFM Waveforms, SNR 12 dB

5.5 EVALUATION

Table 20 through Table 22 show results for the 9+1000, 9+500, 9+100 Random N-Class anomaly detectors with SNR levels spanning 4 to 14 dB. The performance metric definitions are the same as defined in section 4.5.

Table 20. Performance of 9+1000 Random Anomaly Detectors

CNN	AlexNet	GoogleNet	InceptionV3	ResNet50
TNR	87.65	78.24	83.82	85.59
TPR	47.93	91.01	97.96	92.62
PPV	28.68	67.51	90.76	73.49
FPR	12.35	21.77	16.18	14.41
ACC	55.59	88.54	95.24	91.27

Table 21. Performance of 9+500 Random Anomaly Detectors

CNN	AlexNet	GoogleNet	InceptionV3	ResNet50
TNR	71.43	86.31	82.44	93.75
TPR	53.41	93.68	97.96	91.85
PPV	26.58	76.32	90.52	73.09
FPR	28.57	13.69	17.56	06.25
ACC	56.85	92.27	95.00	92.21

Table 22. Performance of 9+100 Random Anomaly Detectors

CNN	AlexNet	GoogleNet	InceptionV3	ResNet50
TNR	82.74	25.89	80.06	00.0
TPR	86.51	97.75	97.96	93.32
PPV	59.15	73.11	90.27	00.0
FPR	17.26	74.11	19.94	100.0
ACC	85.79	84.03	94.54	75.50

For AlexNet with the 9+1000 classes, the overall accuracy has poor performance at 56%. Although the anomaly detection accuracy rate TNR may be acceptable at 88%, TPR indicates that only 48% of the knowns are accepted for classification, implying that the other 52% of knowns are declared as anomalies. This is not acceptable and significantly degrades overall classification performance. With the 9+500 configuration, the overall performance is still unacceptable, with TNR degrading to 71%, TPR slightly improving to 53%, and overall unacceptable accuracy at 56%. Further reduction to the 9+100 configuration counterintuitively shows significant improvement, with TNR at 83%, TPR at 87%, and overall accuracy of 86%. This improvement is hypothesized to be due to fewer nodes colliding with the feature space defined for the known classes.

Following similar reasoning, the AlexNet optimum model is 9+100, the GoogleNet optimum model is 9+500, the InceptionV3 optimum model is 9+1000, and the ResNet50 optimum model is 9+500. Picking the optimal model can be done by selecting the model with the highest ACC value.

5.6 SUMMARY

As one might expect, the results are quite mixed, particularly among the different CNNs. Performance is degraded or improved by increasing or decreasing the number of random nodes, or by randomly generating a separate set of weights and biases for the nodes. Although not reported in these results, experiments showed that the overall performance has a dependency on the initialization seed used for the random number generator. It is hypothesized that the variations of results among the CNNs is primarily due to the very nature of randomly assigning the weights and biases and the feature space of the CNNs themselves. Further experimentation with different random assignments would yield different sets of parameters with acceptable results for all CNNs.

It is speculated that increasing the number of random nodes too high may lead to overall degradation in performance because random assignments are more likely to fall into the space of the known classes as more nodes are generated.

This experiment demonstrated an approach for anomaly detection using M-class perceptron or softmax nodes with randomly generated weights and biases. However, this approach is not practical in that it does not provide for a reliable method to on-the-fly detect and cluster anomalous classes of data input to the classifiers. Therefore, the initial value of the weight and bias parameters must be methodically chosen to be more representative of the anomalous classes as they are detected while online. The experiments carried out in this section have paved the way for the remaining work focused on adaptive classifiers utilizing one-shot learning, centroiding, and clustering.

6 ADAPTIVE CLASSIFIERS

The ability of RF devices to adaptively classify activity in nonstationary environments is a crucial component of cognitive RF systems, therefore the purpose of this study is to explore approaches for offline training of classifiers on known classes and to augment them with adaptive algorithms for online unsupervised waveform detection, clustering, and classification.

6.1 CHALLENGES

An important component of a cognitive RF communication system is the ability to accurately classify multiple classes of waveforms acquired by its sensors. Such classifiers are typically trained using supervised learning techniques on known data sets. However, when these classifiers are presented with unknown waveform classes it was not previously trained on, it will fail. The challenge is that while in the online operation mode, to not only classify the known waveform classes, but to also recognize the presence of anomalous waveforms and to adaptively cluster them into unknown or new subclasses.

6.2 CONTRIBUTIONS

This work demonstrates a unique combination of supervised and unsupervised learning adaptive classifiers for RF waveform classification by presenting the results of four novel online adaptive classifiers that can effectively maintain high accuracy of classification performance in nonstationary RF environments. The novel adaptive classifiers are: 1) Centroid with Anomaly Detection and Clustering (CADC) algorithm; 2) Frequency Hits Anomaly Detection and Clustering (FHITS); 3) DYNG Extended with CADC (DYNG-CADC); and 4) DYNG extended with FHITS (DYNG-FHITS).

Key contributions of this research are the development of the online adaptive learning algorithms CADC and FHITS by combining the concepts of prior knowledge, centroiding, distance measures, and one-shot learning. These two algorithms were then used to enhance the online labeling strategy of the DYNG algorithm.

Finally, a key contribution of these algorithms is that they can be used to make static classifiers such as CNN architectures adaptable by replacing the last classification layers with these adaptive classifiers.

The remainder of this section will discuss these algorithms in detail.

6.3 METHODOLOGY

For each of the four adaptive classifier algorithms, the following are defined in the following subsections in great detail: the specific model; the model's training phase algorithm; the online prediction algorithm, the anomaly detection and adaptation process; and finally, the anomaly insertion operations.

6.3.1 CADC Algorithm

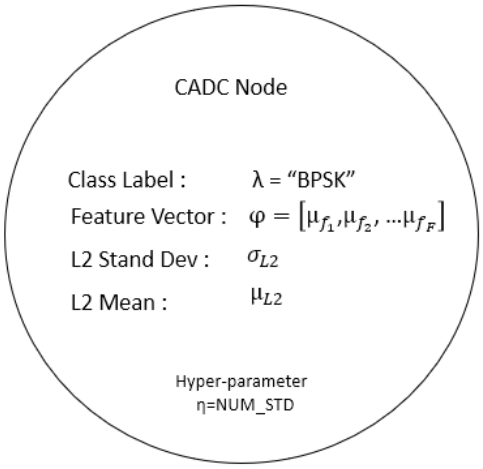


Figure 100. CADC Centroid Node

The CADC algorithm is designed to perform class predictions, anomaly detections, and semi-supervised clustering (adaptation) functions. For each class, the CADC algorithm generates a prototype node as shown in Figure 100. The CADC prototypes have four attributes. The class label λ which defines the name of the class (for example "BPSK"). φ is a feature vector of length F, and considered as the centroid of all training samples from the same class, where each component of φ defines the mean of each corresponding feature from all training data of the class. The value of F is defined by the length of the CNN output layer from which the centroid node takes input. The L2 standard deviation σ_{L2} and mean μ_{L2} establish boundaries on the use of the centroid feature vector and plays a significant role in prediction decisions. The remaining subsection discusses the CADC algorithm in greater detail.

6.3.1.1 Training

The overall purpose of the training process is to establish a CADC centroid model for each class defined in the training data. The CADC algorithm begins with generating what is defined as

labeled centroid nodes. Centroid nodes are defined as having a feature vector, class label, a standard deviation, and mean parameter. The CADC training algorithm pseudocode is shown in Table 23.

Before the CADC can be taken online to perform the tasks of classification, anomaly detection, and anomaly clustering, it must first be trained on the C number of known training classes. After training on the known data set, the CADC will be composed of C centroid nodes, where each centroid node has a unique label λ derived from the labeled training data, and it will have a feature vector φ. The feature vector φ for each centroid is the result of averaging the values at the corresponding feature index positions from all training samples in the same class. Once C prototype centroids are established, the statistics σ_{L2} and μ_{L2} are computed. This is done by reiterating through the training data for each class and calculating the mean and the standard deviation of the L2 distances between a class centroid and all training samples with the same class label. This step provides for statistical boundaries for making classification decisions during predictions. Lastly, all centroids and associated statistics for the known classes are defined and the CADC algorithm can be taken online to perform predictions, anomaly detection, and adaptation functions.

Table 23. CADC Training Algorithm

1. Start with CADC empty node set $S = \{\}$
2. $k = 0$ //initialize selected node index
3. $C = 0$ //initialize number of classes
4. Set Stream = TrainingSet
5. //from training data, compute the centroid vectors for each class
6. While Stream.hasNext do
7. $x := Stream.next$ with $x \in R^N$
8. label= $l(x)$ //get class label from input training vector
9. if classLabelNotInNodeSet(label,S)
10. $C++$
11. $k = C$
12. Create new node n_k
13. $n_k.\lambda$= label
14. $S = \{S, n_k\}$ //add n_k to set S
15. else
16. //return index of the node with the label.
17. $k = getNodeIndex(label, S)$
18. end if
19. //compute the centroid as the running mean of feature vector components
20. for f:=1..F
21. $n_k.\varphi(f) = runningMean(n_k.\varphi(f), x(f))$
22. end for
23. end while
24. // rescan training data to compute the L2 stddev and mean for each class
25. Reset Stream = TrainingSet
26. While Stream.hasNext do
27. $x := Stream.next$ with $x \in R^F$
28. label= $l(x)$ //get label of input vector
29. $k = getNodeIndex(label, S)$
30. //compute the L2 mean and stddev
31. $cost = L2cost(n_k.\varphi, x)$
32. $n_k.\sigma_{L2} = runningSigma(n_k.\sigma_{L2}, cost)$
33. $n_k.\mu_{L2} = runningMean(n_k.\mu_{L2}, cost)$
34. end while
35. //when done training S will contain …
36. $S = \{n_1, n_2, \ldots n_C\}$

6.3.1.2 Online Prediction and Adaptation

With the trained CADC model taken online, it can perform class predictions and adaptations when anomalies are detected. Pseudocode for the CADC online prediction and adaption algorithm is shown in Table 24. The adapt sigma flag is set to true or false, to determine if an anomaly centroid's standard deviation will adapt or be kept frozen as new stimuli are assigned to it (test results report on both cases). As a stimulus is presented to the centroid network, the L2 distance is computed between the input stimulus and all centroid nodes. The centroid with the smallest L2 distance is selected as the predicted candidate class. To confirm acceptance of the

candidate, the number of standard deviations hyper-parameter (η) is used. The L2 distance is checked to determine if the stimulus falls within η standard deviations of the centroids (η=3 used for reported results). If the stimulus' L2 distance falls within the thresholds, it is accepted as classified. If the stimulus is successfully declared as one of the known classes, that prediction stands, and no further processing is required for that stimulus. However, if the stimulus is successfully predicted as one of the unknown classes (as discussed in the next section), the L2 standard deviation and mean statistic values for that unknown class are updated. This action provides for online refinement of the unknown class statistics each time a stimulus is declared a particular unknown class. If the incoming stimulus prediction is rejected because the calculated L2 distance falls outside of the selected centroid's threshold, the stimulus is declared and processed as an anomaly as discussed in the next section.

Table 24. CADC Online Prediction and Adaptation Algorithm

1.	Start with the CADC node set $S = \{n_1, n_2 \dots n_C\}$, *initialized in training step*
2.	Set Stream = OnlineDataStream
3.	Set adaptSigma = {true or false} //true - anomaly centroids sigma's will adapt as new stimuli arrive
4.	//compute the centroid vectors for each class
5.	While Stream.hasNext do
6.	
7.	$costMin = \infty$
8.	$predict = undefined$
9.	$x := Stream.next$ with $x \in R^F$
10.	
11.	//find the candidate prediction
12.	for k:=1:C
13.	$cost = L2cost(n_k.\varphi, x)$
14.	if cost < costMin
15.	$costMin = cost$
16.	$predict$ = k
17.	end if
18.	end for
19.	
20.	//if x is declared an anomaly using the class stats, add the anomaly to the model and report it.
21.	if $(costMin > (n_k.\mu_{L2} + \eta * n_k.\sigma_{L2}))$ *or* $(costMin < (n_k.\mu_{L2} - \eta * n_k.\sigma_{L2}))$
22.	$newUnknownLabel = addAnomallyToModel(x)$
23.	report: prediction = $newUnknownLabel$
24.	continue; //continue to next input stream on while loop
25.	end if
26.	
27.	//since x is not a **new** anomaly, check to see if the prediction is of an //**unknown** class. If it is an unknown class, update the **unknown** // class's model statistics.
28.	if isUnknownClass(S,predict) == true
29.	//compute the running mean of feature vector components
30.	for f:=1..F
31.	$n_{predict}.\varphi(f) = runningMean(n_{predict}.\varphi(f), x(f))$
32.	end for
33.	//compute the running L2 mean and stddev
34.	$cost = L2cost(n_k.\varphi, x)$

```
35.          if adaptSigma==true
36.            n_predict.σ_L2 = runningSigma(n_predict.σ_L2, cost)
37.          end if
38.          n_predict.μ_L2 = runningMean(n_predict.μ_L2, cost)
39.       end if
40.       //report on the class prediction
41.       report: predict
42.    end while
```

6.3.1.3 Anomaly-Class Insertion and Adaptation

The insertion of unknown stimuli into the model is a critical step of the adaptive learning process. Table 25 shows the pseudo code for this process, and it is called the $addAnomallyToModel(x_i)$ function in Table 24. When a stimulus is declared an "anomaly" the goals are to quickly: 1) create a new unknown centroid class; 2) establish prediction statistics for the new unknown class; and 3) report the anomaly as a new unknown class. The approach relies on the concept of one-shot learning, and the current statistics of all nodes of the CADC model. When the first anomaly is detected, a new centroid class is created and labelled as "unknown1", and when a second anomaly is detected the new centroid class is created and labelled as "unknown2", etc. The feature vector of the new unknown centroid is initialized to the value of the stimulus vector. Then, to establish a model on the new class, the statistical knowledge already established on the centroids presently in the network model is leveraged. In doing this, the L2 standard deviation and mean for the new centroid is initialized to be the average of all the L2 standard deviations and means of all the existing centroid nodes. This average includes averaging the statistics from the known and unknown classes. Upon insertion completion, a new unknown node has been generated with a reasonable starting vector and statistics.

Table 25. CADC Anomaly Insertion and Adaptation Algorithm

1.	Start with CADC model: $S = \{n_1, n_2 \dots n_C\}$, and anomaly vector x
2.	//C is the number of present clusters
3.	$k = C + 1$
4.	Create a new CADC node n_k
5.	//Create a new unknown label = "unknown#", where # is the new label's unique index number.
6.	$n_k.label =$ "unknown#"
7.	//assign nodes feature vector the stimulus (one shot learning)
8.	$n_k.\varphi = x$
9.	//for all CADC nodes in the model, compute the mean of their stddev and means.
10.	//Use those values for n_k's initial statistics.
11.	$for\ c = 1{:}\ C$
12.	$n_k.\sigma_{L2} = runningMean(n_c.\sigma_{L2}, n_k.\sigma_{L2})$
13.	$n_k.\mu_{L2} = runningMean(n_c.\mu_{L2}, n_k.\mu_{L2})$
14.	$end\ for$
15.	$S = \{S, n_k\} = = \{n_1, n_2 \dots n_{C,} n_k\}$
16.	$C++$

6.3.2 FHITS Algorithm

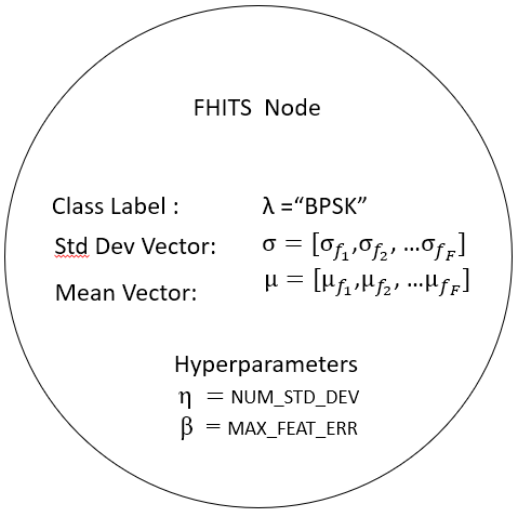


Figure 101. FHITS

For each class, the FHITS algorithm generates a prototype node as shown in Figure 101. The class label λ defines the name of the class (for example "BPSK"). The standard deviation (σ) and mean (μ) vectors establish decision boundaries for predictions. FHITS also requires two hyper-parameters, η and β. The integer η defines the number of standard deviations a feature vector's components can fall outside of the boundary before an error is declared. The real number β that can

take on any value from 1.0 to 0 defines the percent of feature vector components that can be in error before a stimulus is declared an anomaly. Further discussion of the initialization, calculations, online modifications, and use of these parameters is detailed in the pseudo code in the remaining sub sections.

6.3.2.1 Training

Pseudo code for the FHITS training algorithm is shown in Table 26. The algorithm begins with generating labelled centroid nodes, n_k. The nodes have a class label, a standard deviation feature vector (σ), and a mean feature vector (μ). The feature vector dimension is defined by the length of the CNN output layer that the centroid node will take input from (this size is F). Before FHITS can be taken online, it must first be trained on the C known classes. Thereafter, the FHITS model will consist of C nodes. The μ feature vector is calculated similarly to CADC. It is the result of averaging the values at the corresponding feature index positions from all x_i training samples within the same class. The σ feature vector is the result of calculating the standard deviation at the corresponding feature index positions from all x_i training samples within the same class.

Table 26. FHITS Training Algorithm

```
Start with FHITS empty node set S = {}
k = 0   //initialize selected node index
C = 0   //initialize number of classes
Set Stream = TrainingSet
//from training data, compute the statistic vectors for each class
While Stream.hasNext do
    x := Stream.next with x ∈ R^F
    label= l(x)  //get class label from input training vector
    if  classLabelNotInNodeSet(label,S)
        C++
        k = C
        Create new node n_k
        n_k.λ= label
        S = {S,  n_k} //add n_k to set S
    else
        //return index of the node with the label.
        k = getNodeIndex(label, S)
    end if
    //compute the mean and standard deviation feature vectors
    for f: =1..F
      n_k.μ(f) = runningMean(n_k.μ(f), x(f) )
```

	22. $n_k.\sigma(f) = runningMean(n_k.\sigma(f), x(f))$ 23. end for 24. end while 25. //when done training S will contain … 26. $S = \{n_1, n_2, \ldots n_c\}$

6.3.2.2 Online Prediction and Adaptation

With the trained network online, it can perform class predictions, anomaly detection, and adaptation. Table 27 shows the pseudo code for FHITS Online Prediction and Adaption algorithm. The adaptSigma flag is set to true or false, to determine if an anomaly centroid's standard deviation will adapt or be kept frozen as new stimuli are assigned to it (test results report on both cases). As an x stimuli is presented to the network, the frequency of error hits parameter $errorCount$ is computed between the input stimulus and all class nodes in the network. To carry out this computation, for all class centroids n_k, each component of the stimulus vector ($x[f]$) is checked to determine if it falls within $\pm\eta$ standard deviations of the component's average using the statistics computed during training. If the stimulus vector falls outside the statistical limits, $errorCount$ is increased therefore indicating a non-match. The class with the least hits is a candidate for the predicted class of the stimulus.

To confirm acceptance of the class prediction, the hyper parameter threshold β is used. The β parameter defines the maximum percentage of error hits allowed before the stimulus is declared an anomaly. If the stimulus' hit counter falls below the threshold, the stimulus is accepted as classified. If the stimulus is successfully declared as one of the known classes defined in the training set, then the prediction is accepted. However, if the stimulus is predicted as an unknown class (as discussed in the next section), the mean statistic values are updated using the stimulus value. This action provides for online refinement of the unknown class statistics each time a stimulus is

declared as a particular unknown class. If the incoming stimulus prediction is rejected because the calculated hit counter falls above the threshold, the stimulus is declared as an anomaly and insertion processed as discussed in the next section.

Table 27. FHITS Online Prediction and Adaptation Algorithm

1.	Start with the FHITS node set $S = \{n_1, n_2 \dots n_C\}$, *initialized in training step*
2.	Set Stream = OnlineDataStream
3.	Set β
4.	//compute the statistics vectors for each class
5.	While Stream.hasNext do
6.	
7.	$errorCount = 0$
8.	$minErrorCount = \infty$
9.	$predict = undefined$
10.	
11.	$x := Stream.next$ with $x \in R^F$
12.	
13.	//find the candidate prediction
14.	for k:=1:C
15.	$errorCount = 0$
16.	for f=1:F
17.	if $(x[f] < (n_k.\mu[f] - \eta * n_k.\sigma[f]))$ *or* $(x[f] > (n_k.\mu[f] + \eta * n_k.\sigma[f]))$
18.	$errorCount++$
19.	end if
20.	end for
21.	
22.	//select class with lowest error count
23.	if $errorCount < minErrorCount$
24.	$minErrorCount = errorCount$
25.	$predict =$ k
26.	end if
27.	end for
28.	
29.	//if x_i is declared an anomaly, add the anomaly to the model and report it.
30.	if $(minErrorCount > \beta * F)$
31.	$newUnknownLabel = addAnomallyToModel(S, x)$
32.	report: prediction = $newUnknownLabel$
33.	continue; //continue to next input stream on main while loop
34.	end if
35.	
36.	//since x is not a **new** anomaly, check to see if the prediction is of an **unknown**
37.	//class. If it is an unknown class, update the **unknown** class's model statistics.
38.	if isUnknownClass(S, predict) == true
39.	//compute the mean of feature vector components
40.	for f:=1..F
41.	$n_{predict}.\mu(f) = runningMean(n_{predict}.\mu(f), x(f))$
42.	if adaptSigma
43.	$n_{predict}.\sigma(f) = runningMean(n_{predict}.\sigma(f), x(f))$
44.	end if
45.	end for
46.	end if
47.	
48.	//report on the class prediction
49.	report: predict
50.	end while

6.3.2.3 Anomaly-Class Insertion and Adaptation

When a stimulus is declared an "anomaly" the goals are to: 1) create a new unknown node class; 2) quickly establish prediction statistics for the new unknown class; and 3) report the anomaly as a new unknown class. Table 28 shows the pseudo code for these operations. To create unknown centroid classes, a running index counter is used starting at value 1. When the first anomaly is detected, a new centroid class is created and labelled as "unknown1", and when a second anomaly is detected a new centroid class is created and labelled as "unknown2", etc. To quickly establish the centroid statistics, the feature vector of the new unknown centroid is initialized to the value of the stimulus vector. Then to establish a statistical model of the new class, the knowledge already established on the centroids presently in the network model is leveraged by assigning the standard deviation and mean vector of the new centroid to be the average of all the standard deviations and means of the statistics of all the existing centroid classes. This includes averaging the statistics from the known and unknown classes.

Table 28. FHITS Anomaly Insertion and Adaptation Algorithm

	1. Start with the FHITS model: $S = \{n_1, n_2 \dots n_C\}$, and anomaly vector x
	2. $k = C + 1$
	3. Create a new node n_k
	4. //Create a new unknown label = "unknown#",
	//where # is the new label's unique index number.
	5. $n_k.label =$ "unknown#"
	6. $n_k.\mu = x$ //set mean vector to the stimulus (one shot learning)
	7. //for all FHITS nodes in the model,
	//compute the mean of their feature components' stddev .
	8. //Use those values for $n_k.\sigma$'s initial statistics.
	9. $for\ f = 1..F$
	10. for c=1..C
	11. $n_k.\sigma(f) = runningMean(n_c.\sigma(f)\ ,\ n_k.\sigma(f)\ )$
	12. end for
	13. $end\ for$
	14. $S = \{S, n_k\} = \{n_1, n_2 \dots n_{c,} n_k\}$
	15. $C++$

6.3.3 DYNG-CADC Algorithm

There is great interest in exploring the capabilities of DYNG for online classification of streaming data because of its great flexibility in adapting to non-stationary data. DYNG does not provide a mechanism for anomaly detection and on the fly labeling of detected anomalies, and therefore CADC will play such a role by combining CADC with DYNG.

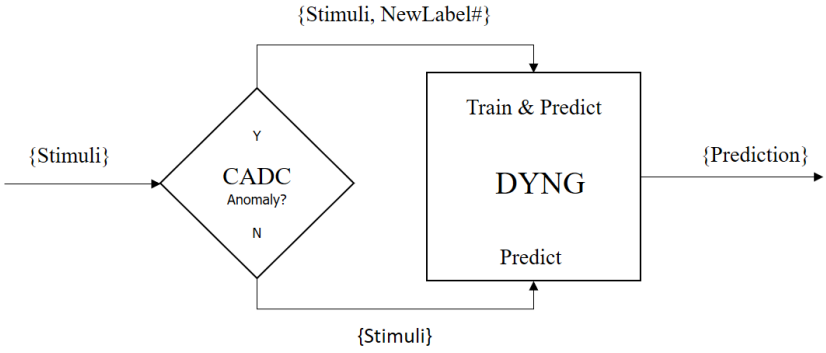


Figure 102. DYNG-CADC Architecture

Figure 102 shows the DYNG-CADC architecture, where the strategy of DYNG-CADC is to extend (or augment) DYNG with CADC to allow for rapid online anomaly detection and unique labelling of new anomaly stimulus classes as they are input to the architecture. CADC will act as an adaptive unsupervised online: anomaly detector; anomaly labeler; and training controller for DYNG. CADC in effect dynamically manages the unsupervised training and growth of DYNG. The remainder of this subsection discusses the complete details of the DYNG-CADC in operation.

6.3.3.1 Training

Before the DYNG-CADC algorithm is taken online, both DYNG and CADC must first be trained on C known classes of training data. A complete description of the training algorithm for DYNG is provided in [106]. After training, the initial DYNG model is composed of C clusters of nodes representing the C known classes. The number of nodes within each cluster varies per class and is determined dynamically by the DYNG training algorithm. The CADC network structure is also created and trained on the training data set as described in the CADC training section. In the DYNG-CADC algorithm, the CADC structure is only used as an online adaptive anomaly detector, and the DYNG network is used for classification. It is expected (not proven) that the DYNG network will provide greater classification accuracy than CADC because it has multiple nodes representing each class (therefore more voting power) than CADC, which has only one node structure per class. The hyper-parameters for the DYNG networks are shown in Table 29. MAX_NODES was set to 100 nodes to limit the growth of the DYNG networks. That yields approximately 11 DYNG nodes for each of the 9 training classes. The MAX_AGE was set relatively low to minimize growth of the network.

Table 29. DYNG-CADC Hyperparameters

Parameter	Value
EB	0.1
EN	0.0006
ALPHA	0.5
D	0.0005
LAMBDA	30
MAX_AGE	25
MAX_NODES	100
NUM_STD	5.0

6.3.3.2 Online Prediction and Adaptation

Once DYNG-CADC is trained on the known data set, it can be taken online for classification, anomaly detection, and clustering of unknown stimuli. In this mode, CADC processes the stimuli first to determine if the stimuli is declared an anomaly. If not declared as such, DYNG is used to perform the stimuli classification. If CADC declares the stimuli an anomaly, a unique label is generated and the stimuli with the new label is input to the DYNG and CADC in one-shot training modes to incorporate the new class into the model. See section 6.3.1 for more details on the CADC algorithm, and [106] for more details on DYNG.

Table 30. DYNG-CADC Algorithm

```
1.  Start with the DYNG-CADC model, initialized in training step
2.  Set Stream = OnlineDataStream
3.  //compute the statistics vectors for each class
4.  While Stream.hasNext do
5.     x_i := Stream.next(t) with x_i ∈ R^n
6.
7.     costMin = ∞
8.     predict = undefined
9.     x_i := Stream.next(t) with x_i ∈ R^N
10.    //find the candidate prediction
11.    for k:=1:C
12.       cost = L2cost(n_k.φ, x_i)
13.       if cost < costMin
14.          costMin = cost
15.          predict = k
16.       end if
17.    end for
18.
19. //if x_i is declared an anomaly using the class stats,
    //add the anomaly to the model and report it.
20.    if (costMin > (n_k.μ_L2 + η * n_k.σ_L2)) or
              (costMin < (n_k.μ_L2 − η * n_k.σ_L2))
21.       //add new anomaly to CADC model
22.       newUnknownLabel = addAnomallyToModel(x_i)
23.       prediction = newUnknownLabel
24.       //train DYNG model on new anomaly – one shot
25.       trainDYNG (x_i, newUnknownLabel)
26.       continue; //continue to next input stream on while loop
27.    end if
28.
29.    // use DYNG model to make the actual class prediction of x_i
30.    prediction = predictDYNG (x_i);
31.
32.    //since x_i is not a new anomaly, check to see if the prediction is of an unknown
33.    //class. If it is an unknown class, update the unknown class's model statistics.
34.    if isUnknownClass(S,predict) == true
35.       //compute the running mean of feature vector components
36.       for f:=1..N
```

```
37.            n_k.φ(f) = runningMean(n_k.φ(f), x_i(f) )
38.          end for
39.          //compute the running L2 mean and stddev
40.          cost  = L2cost(n_k.φ, x_i )
41.          n_k.σ_L2 = runningSigma(n_k.σ_L2, cost)
42.          n_k.μ_L2 = runningMean(n_k.μ_L2, cost)
43.
44.           //update DYNG
45.           trainDYNG (prediction)
46.
47.        end if
48.
49.     //report on the class prediction
50.     report : prediction
51.     t++
52.  end while
```

6.3.3.3 Anomaly-Class Insertion and Adaptation

The Anomaly Insertion and Adaption into the CADC data structure is performed as described in section 6.3.1.3.

6.3.4 DYNG-FHITS Algorithm

There is great interest in this research in exploring the capabilities of DYNG for online classification of streaming data because of its great flexibility in adapting to non-stationary data. However, because DYNG does not provide a mechanism for anomaly detection and on the fly labeling of detected anomalies, FHITS will play such a role by combining it with DYNG. Figure 103 shows the derived DYNG-FHITS architecture.

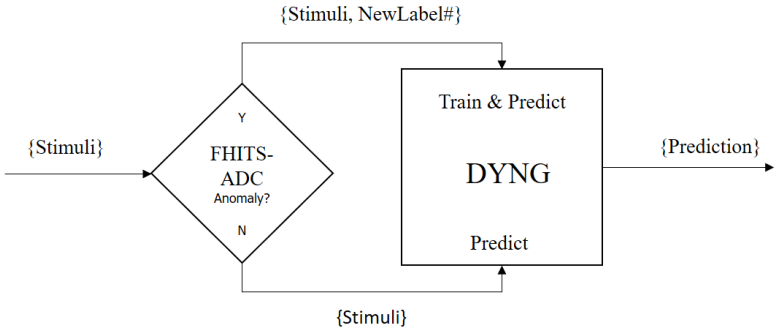


Figure 103. DYNG-FHITS Architecture

The strategy is to extend DYNG with FHITS to allow for rapid online anomaly detection and automatic labelling of new anomaly stimulus classes as they are input to the architecture. FHITS acts as an online adaptive unsupervised: anomaly detector; anomaly labeler; and controls the DYNG online training process. The remainder of this subsection discusses the complete details of DYNG- FHITS.

6.3.4.1 Training

Before the DYNG-FHITS algorithm is taken online, both DYNG and FHITS must be trained on C number of known classes of training data. A complete description of the training algorithm for DYNG is provided in [106]. The FHITS structure with C nodes is created as described in the FHITS training section 6.3.2.1. After DYNG training, the model is composed of C clusters of nodes representing the C known classes. The number of nodes within each DYNG cluster varies per class and is determined dynamically by the DYNG training algorithm. As depicted in Figure 103 for the DYNG-FHITS algorithm, FHITS is only used as an online adaptive anomaly detector and labeler, and the DYNG network is used for classification. The hyper-parameters for the DYNG networks are shown in Table 31 . MAX_NODES is set to 100 nodes to limit the growth of the DYNG networks. That yields approximately 11 DYNG nodes for each of the 9 training classes. The MAX_AGE is set relatively low to also minimize growth of the network.

Table 31. DYNG Hyperparameters

Parameter	Value
EB	0.1
EN	0.0006
ALPHA	0.5
D	0.0005
LAMBDA	30
MAX_AGE	25
NUM_STD	5.0

6.3.4.2 Online Prediction and Adaptation

Table 32 shows pseudo code for the DYNG-FHITS online prediction and adaptation algorithm. Once DYNG-FHITS is trained on the known data set, it can be taken online for classification, anomaly detection, and clustering of unknown stimuli. In this mode, FHITS processes the stimuli first to determine if the stimuli are declared an anomaly. If not declared as such, DYNG is used to perform the stimuli classification. If FHITS declares the stimuli an anomaly, a unique label is generated and the stimuli with the new label is input to DYNG and FHITS in a one-shot training mode to incorporate the new class into the architectures. See section 6.3.2 for more details on the FHITS, and [106] for more details on the DYNG.

Table 32. DYNG-FHITS Algorithm

1. Start with the FHITS node set $S = \{n_1, n_2 \dots n_C\}$, *initialized in training step*
2. Start with the DYNG structure, initialized *in training step*
3.
4. Set Stream = OnlineDataStream
5. //compute the statistics vectors for each class
6. While Stream.hasNext do
7.
8. $errorCount = 0$
9. $minErrorCount = \infty$
10. $predict = undefined$
11.
12. $x_i := Stream.next(t)$ with $x_i \in R^N$
13.
14. //find the candidate prediction
15. for k:=1:C
16. $errorCount = 0$
17. for f=1:N
18. if $(x_i[f] < (n_k.\mu[f] - \eta * n_k.\sigma[f]))$ *or* $(x_i[f] > (n_k.\mu[f] + \eta * n_k.\sigma[f]))$

```
                errorCount + +
              end if
            end for

            //select class with lowest error count
            if errorCount < minErrorCount
               minErrorCount = errorCount
               predict = k
            end if
          end for

      //if x_i is declared an anomaly, add the anomaly to the model and report it.
          if (minErrorCount > β * N)
            newUnknownLabel = addAnomallyToModel(S, x_i)
            report : prediction = newUnknownLabel
            //train DYNG model on new anomaly – one shot
            trainDYNG (x_i, newUnknownLabel)
            continue; //continue to next input stream on main while loop
          end if

          // use DYNG model to make the actual class prediction of x_i
          prediction = predictDYNG (x_i);
          //since x_i is not a new anomaly, check to see if the prediction is of an unknown
          //class. If it is an unknown class, update the unknown class's model statistics.
          //also update the dyng model on the declared unknown type
          if isUnknownClass(S,predict) == true
            //compute the mean of feature vector components
            for f:=1..N
                n_k.μ(f) = runningMean(n_k.μ(f), x_i(f))
                if adaptSigma
                  n_k.σ(f) = runningMean(n_k.σ(f), x_i(f))
                end if
            end for
           end if

          //update DYNG
          trainDYNG (prediction)

          //report on the class prediction
          report : prediction = k
          t + +
      end while
```

6.3.4.3 Anomaly-Class Insertion and Adaptation

Anomaly Insertion and Adaption into the FHITS data structure is performed as described in section 6.3.2.3.

6.3.5 Complexity Analysis

In the interest of providing computational complexity estimates for the CADC, FHITS, DYNG CADC, and DYNG-FHITS prediction algorithms, a heuristic approach by [107] is used to

summarize the upper bounds on the time and space complexities, with results using the Big-O notation O(…) presented in

Table *33*, **Time and Space Complexities for: CADC, FHITS, DYNG-ADC, and DYNG-FHITS Algorithms**, where C is the number of classes, N is the number of input stimuli, and M is the average number of nodes per class.

Table 33, Time and Space Complexities for:
CADC, FHITS, DYNG-ADC, and DYNG-FHITS Algorithms

Model	Time Complexity	Space Complexity
CADC	*O(NC)*	*O(C)*
FHITS	*O(NC)*	*O(C)*
DYNG-CADC	*O(NMC)*	*O(MC)*
DYNG-FHITS	*O(NMC)*	*O(MC)*

For the CADC and FHITS algorithms, given their data structures with one node per class, the approximate best- and worst-case time complexities is *O(NC)* since all nodes in the models must be searched to find the nearest to each input sample. A space complexity of *O(C)* is required to store the model information for each of the *C* nodes.

For the DYNG-CADC and DYNG-FHITS algorithms, the complexity of the data structures is dominated by the requirements of the DYNG algorithm. There are on average MC number of nodes in the DYNG network, therefore the approximate best- and worst-case time complexities is O(NMC) since all nodes in the model must be searched to find the node nearest to each input

sample. A space complexity of O(MC) is needed to store the model information for the average M nodes per C classes. As seen in

Table **33, Time and Space Complexities for: CADC, FHITS, DYNG-ADC, and DYNG-FHITS Algorithms**, DYNG-CADC and DYNG-FHITS have greatest complexities when M >1, and with M=1, the complexities of all algorithms are the same.

6.4 DATA SETS

To test performance of the adaptive classifiers two datasets were used. The first dataset is a two dimensional (2D) simple dataset synthesized to test the basic performance of the CADC and FHITS algorithms. Once the performance of the algorithms was assessed on the 2D data sets, the more complicated waveform datasets were used to perform more extensive testing on the CNN, CNN-CADC, CNN-FHITS, and CNN-DYNG-FHITS, and CNN-DYNG-CADC implementations. The following two subsections discuss the datasets in greater detail.

6.4.1 ART 2D Datasets

The artificial dataset (ART) was created to preliminarily explore performance of the CADC and FHITS algorithms before they were to be applied to the more complex waveform datasets. The ART dataset is a simple two-dimensional Gaussian mixture distribution with 12 known categories labeled K1 through K12, and 6 anomaly categories (or unknowns) labeled A1 through A6. Using a standard deviation of 1, each category had 50 samples and were generated as shown in Figure 104. Table 34 and Table 35 show the centroids of the known and anomalous classes. The goal of using ART is to gain insight and intuition on their performance limitations. As depicted in Figure

104, the anomaly classes (indicated by colored stars) were designed to have slight overlap with their neighboring known classes, but none of the known classes (indicated by solid circles) overlap with each other. This provides a configuration to determine how the adaptive anomaly detection and classification algorithms respond to such scenarios. Also note from Figure 104 that there is minimal overlap between the knowns and neighboring unknowns due to the spacing of the centroids and the tight standard deviations of the populations.

Table 34. ART Known Clusters

Class	X	Y
K1	-10	-5
K2	-10	5
K3	-5	-10
K4	-5	0
K5	-5	10
K6	0	-5
K7	0	5
K8	5	-10
K9	5	0
K10	5	10
K11	10	-5
K12	10	5

Table 35. ART Anomaly Clusters

Class	X	Y
A1	-10	0
A2	-5	-5
A3	-5	5
A4	0	0
A5	5	-5
A6	5	5

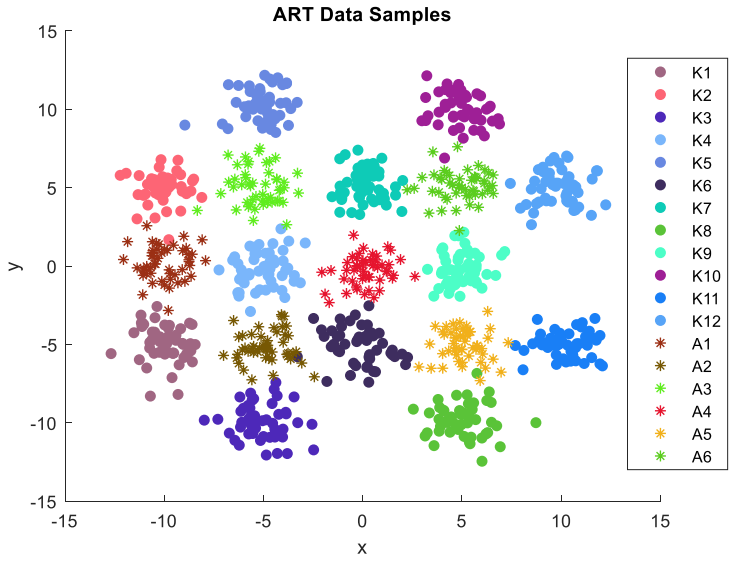


Figure 104. ART 2D Dataset

6.4.2 RF Waveform Datasets

To test performance of the CNN, CNN-CADC, CNN-FHITS, and CNN-DYNG-FHITS, and CNN-DYNG-CADC implementations, the known waveform datasets described in section 3.4 and Table 9, and the anomalous waveform datasets described in section 4.4.2 and Table 11 were used. The detailed procedure is provided in Appendix A. that discusses how the known and anomalous datasets were sampled and then input to the algorithms, to mimic real streaming data scenarios to assess the adaptive classifiers.

6.5 EVALUATION

The results of the CADC, FHITS, and DYNG-CADC algorithms are presented. There are two evaluation subsections. The first subsection 6.5.1 provides results for the CADC and FHITS algorithms applied to the simple 2D ART datasets. The second subsection 6.5.1.3 provides results for the CADC, FHITS, DYNG-CADC, and the DYNG-FHITS algorithms applied to the RF waveform datasets passed through the CNNs used as feature extractors.

Recall for the CADC and FHITS adaptive algorithms discussed in section 6, that there are two types of adaptive parameters for implementing the fly modeling of the anomalous classes. The first parameter type is the mean centroids, and the second parameter type is the sigma values used to establish threshold boundaries about the means. Recall also that the algorithms can be run in one of two modes, with the first mode referred to as *adaptive-sigma*, and is when anomalous stimuli are detected and classified into a specific anomaly class, the mean and sigma parameters of the algorithm are permitted to adapt. The second mode is referred to as *σ-freeze*, and is when anomalous

stimuli are detected and classified into a specific anomaly class, and the mean is permitted to adapt but the sigma parameters are kept frozen at their initially assigned values. These modes were explored to determine which modes might provide better performance.

The results are reported with respect to the number of positives (p) defined as the number of known test samples, and the number of negatives (n) defined as the number of anomaly samples. The definitions below summarize acronyms used in reporting results.

- (n) is an integer defining the number of anomaly samples
- (p) is an integer defining the number of known samples
- True positives (TP) are the number of correctly identified knowns.
- True negatives (TN) are the number of correctly identified anomalies (unknowns).
- False positives (FP) are the number of incorrectly identified knowns.
- False negatives (FN) are the number of incorrectly identified anomalies (unknowns).
- The true positive rate (TPR) is defined as TP/p.
- The true negative rate (TNR) is defined as TN/n.
- The false negative rate (FNR) is defined as FN/n.
- The overall accuracy (ACC) is defined as (TP+TN)/(p+n).
- The number of Principal components retained for the model (NoPCs)
- Non adaptive positive rate (NAPR) - defines the overall accuracy of how the classifier would perform on the known dataset without adaptive capabilities under the conditions that known and anomalous data points are input to the non-adaptive classifier. This benchmark provides a relative baseline on how well the adaptive algorithms perform on the known and anomalous data sets had it not adapted. NAPR is defined as TP/(p+n).
- Percentage of PCA variance (PVAR) resulting from using the number of principal components.

6.5.1 ART Results

Experiments using the ART data were carried out with the CADC and FHITS algorithms to identify any subtleties in their performance limitations and confirm their correct behavior.

6.5.1.1 CADC

The results from this experiment are shown in Table 36 for ART CADC frozen sigma and Table 37 for ART CADC adaptive sigma, with the hyper parameter set to NUM_STD=3.0. The corresponding data results are plotted in Figure 105 with sigma frozen, and Figure 106 with sigma permitted to adapt. As a reminder, even when sigma is frozen, the anomaly centroids' means still adapt when online. When zero anomalies are introduced, the accuracies are at 100% as expected because the known clusters do not overlap, however as the number of anomalies increases, and as expected, some minimal degradation occurs in both modes. This occurs because a percentage of the anomaly categories overlap with the known categories and those samples are linearly inseparable. The inability to separate overlapping stimuli exposes a key weakness of CADC.

Note in Table 36 and Table 37 , as the number of anomalies increases to 5 or 6, the overall performance slightly increases. This is due to a procedural artifact in the way the data samples shown in Figure 104 are chosen to maintain a test ratio of about 50% anomalies to 50% knowns. As the number of anomalies increases, the number of knowns are also increased to maintain the 50% ratio, and this in turn increases the number of known and anomalous samples that don't overlap, therefore slightly increasing the TNR and TPR performance metrics. This explanation can also be noted by the fact that the TPR rate slightly increases and can be observed in Figure 104. Appendix A gives a more detailed explanation of the sampling process for anomaly testing.

Table 36. ART CADC - σ-freeze Results

Anomalies	ACC	TPR	TNR	NAPR
0	100.0	100.0	-	100.0
1	99.0	98.9	99.2	50.0
2	98.7	99.0	98.3	50.0
3	98.3	98.5	98.1	50.0
4	97.0	97.8	96.2	50.0
5	97.7	98.3	97.1	50.0
6	98.3	98.7	97.8	50.0

Table 37. ART CADC – σ-adapt Results

Anomalies	ACC	TPR	TNR	NAPR
0	100.0	100.0	-	100.0
1	99.2	98.9	99.4	50.0
2	98.4	98.1	98.7	50.0
3	98.1	97.6	98.7	50.0
4	97.8	98.7	96.8	50.0
5	96.9	97.9	95.8	50.0
6	97.2	97.6	96.8	50.0

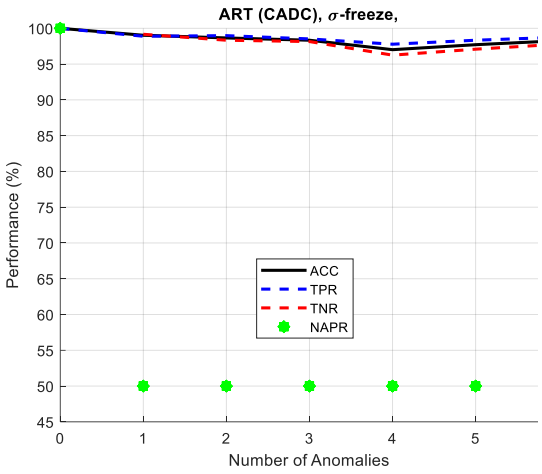


Figure 105. ART CADC σ-freeze

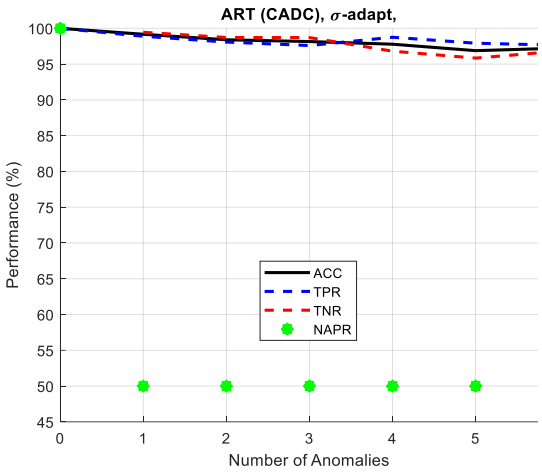


Figure 106. ART CADC σ-adapt

From the figures, one can see that six anomalies were introduced. The green dots indicate the NAPR and shows that for the given dataset if the classifier does not adapt, it would perform at 50% accuracy. This implies that half the data is of the known categories which CADC classifies correctly. The other half of the data are the unknowns which would be classified incorrectly if CADC did not adapt. With online adaptation, CADC can detect and cluster anomalies as they occur in real-time. In comparing the frozen verses adaptive sigma modes, results show that there is no significant performance benefit in using one mode over the other.

6.5.1.2 FHITS

The hyper parameters used for the FHITS configuration were:

NUM_STD 2.5
MAX_FEAT_ERR 0.0

The results are shown in Table 38 for FHITS frozen sigma and Table 39 for FHITS adaptive sigma. The data is plotted in Figure 107 and Figure 108. One can see that when zero anomalies are introduced, the accuracy results are 100% as expected, and as more anomalies are introduced, minimal degradation occurs. The degradation is minimal because only a small percentage of anomalous data overlaps with its known neighboring data samples. Again, the green dots indicate the NAPR and show that for the given dataset if the classifier does not adapt.

Table 38. ART FHITS σ-freeze

Anomalies	ACC	TPR	TNR	NAPR
0	100.0	100.0	NaN	100.0
1	99.0	100.0	98.1	50.0
2	98.8	100.0	97.7	50.0
3	99.1	100.0	98.1	50.0
4	99.3	100.0	98.6	50.0
5	99.0	100.0	97.9	50.0
6	99.0	100.0	97.9	50.0

Table 39. ART FHITS σ-adapt

Anomalies	ACC	TPR	TNR	NAPR
0	100.0	100.0	-	100.0
1	99.0	100.0	98.1	50.0
2	99.2	100.0	98.5	50.0
3	98.8	100.0	97.6	50.0
4	98.9	100.0	97.8	50.0
5	98.7	100.0	97.5	50.0
6	98.9	100.0	97.8	50.0

The results with sigma frozen show a marginal performance benefit over when sigma adapts, which is consistent with the CADC performance. FHITS appears to provide better performance than CADC as the number of anomalies increases, however the differences are minimal.

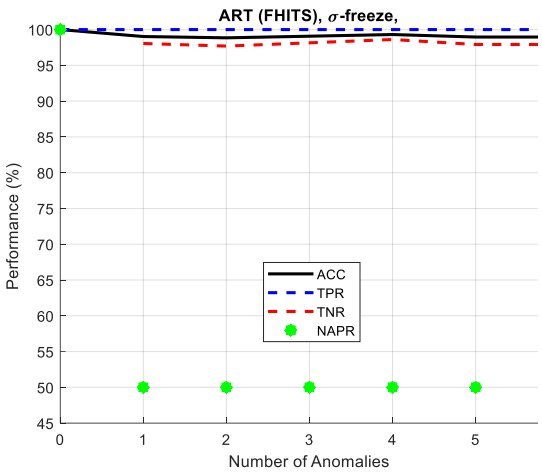


Figure 107. FHITS σ-freeze

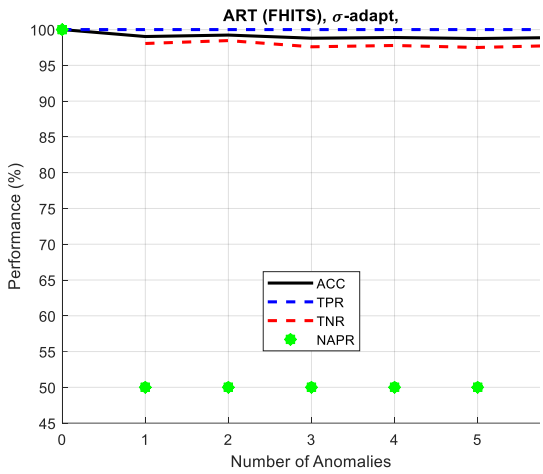


Figure 108. FHITS σ-adapt

6.5.1.3 Comparative Summary

This section presented a preliminary test of the CADC and FHITS algorithms operating in their σ-freeze and σ-adapt modes and tested on the simple 2D ART dataset. Results showed that both algorithms provided acceptable performance on the datasets in both modes, with results indicating that the σ-freeze mode provides marginally better performance. The results also suggest that FHITS provides marginal performance over the CADC algorithm with the ART dataset.

6.5.2 RF Waveform Results

This section provides results on the non-adaptive and adaptive modes of the classifier algorithms. The non-adaptive mode is considered to determine how the algorithms perform when tested with only known waveform classes. The more important and interesting adaptive mode is considered to determine how the algorithms perform when tested with the known and anomalous waveform classes.

6.5.2.1 Nonadaptive Performance

This sub-section reports on the nonadaptive baseline performance accuracies of the CNN classifiers compared to the CNNs augmented with the adaptive classifiers CADC, FHITS, and DYNG. In this series of tests, anomaly detection and on-the-fly clustering functions for the adaptive algorithms are disabled, and only the known datasets are presented to the classifiers. The interest in these series of tests is to compare the original CNN classifiers against the adaptive algorithms to determine if performance between them is consistent. Getting consistent results implies the robustness of the CADC, FHITS, and DYNG algorithms across the different feature vectors generated from the various CNNs.

6.5.2.1.1 CNN

This section discusses performance results of each CNN when presented with the known test data. For the convenience of discussion Figure 109 is a copy of Figure 36 from section 3.5 showing the performance results of the five CNNs. Overall, the networks perform well, and the accuracies are expected to increase at the higher SNR levels, however unexpected drops occur at SNR levels from 6 to 14 for ResNet50, AlexNet, and GoogleNet. It is speculated that these drops occur due to either overfitting on the training data, or due to not training long enough to reach optimal performance. In the case of overfitting the view is that during training, the model over fits the training data and does not generalize enough. Then when performance testing is carried out with the test data (results shown in the plots), the models are presented with new test samples that they were not trained on, and therefore misclassifies. A similar view is speculated in the case of not training the model long enough, therefore it does not generalize enough to accurately classify new test samples. Further training was not pursued because the networks generated acceptable feature vectors for developing and testing the adaptive classifier algorithms (discussed in later sections).

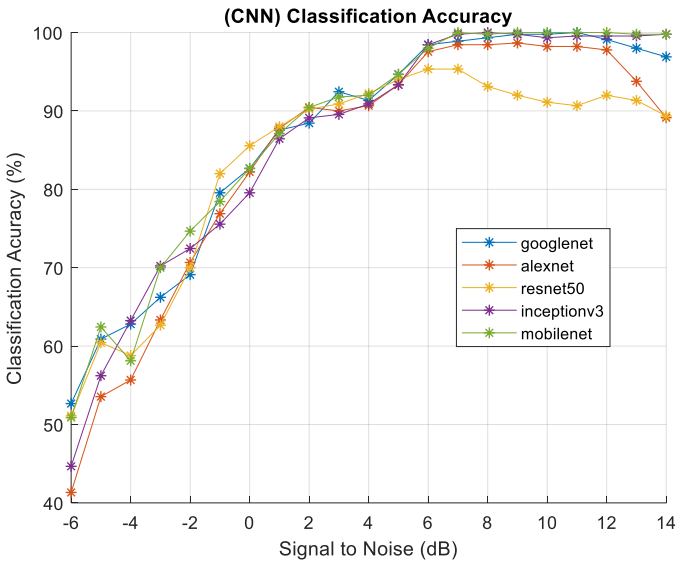


Figure 109. CNN Baseline Accuracy with respect to SNR - Known Datasets

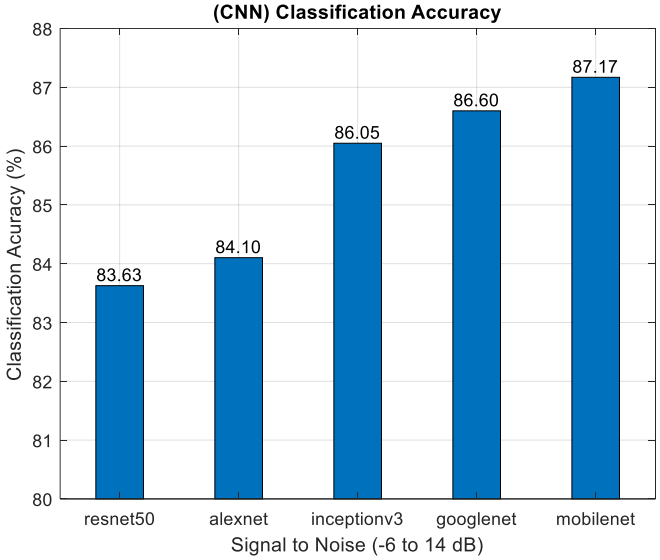


Figure 110. CNN Overall Accuracy - Known Datasets

Figure 110 shows the overall performance accuracy of the CNNs. MobileNet provided the greatest overall performance at about 87%, with resenet50 performing the least accurate. Note that for the purposes of this study, the networks were not trained to give optimal performance on the

datasets, so one network performing greater than another in this research is insignificant. With further training the relative performance gaps may decrease.

6.5.2.1.2 CNN-CADC

This section discusses the performance results of each of the CNN having its last layer replaced with the CADC algorithm operating in its non-adaptive mode. Figure 111 shows the performance results of the five CNN-CADC networks. Overall, the networks perform well. The accuracy of the classifiers increases as the SNR levels increase. An unexpected observation for high SNR levels from 6 to 14 is that the ResNet50 tends to drop, while AlexNet drops at about 9 dB but recovers as SNR increases. This is to the behavior observed in the CNN baseline performance in section 6.5.2.1.1 where the three networks GoogleNet, AlexNet, and ResNet50 dropped in performance in the SNR range. Further investigation is warranted to determine the cause differences, but not explored in this study.

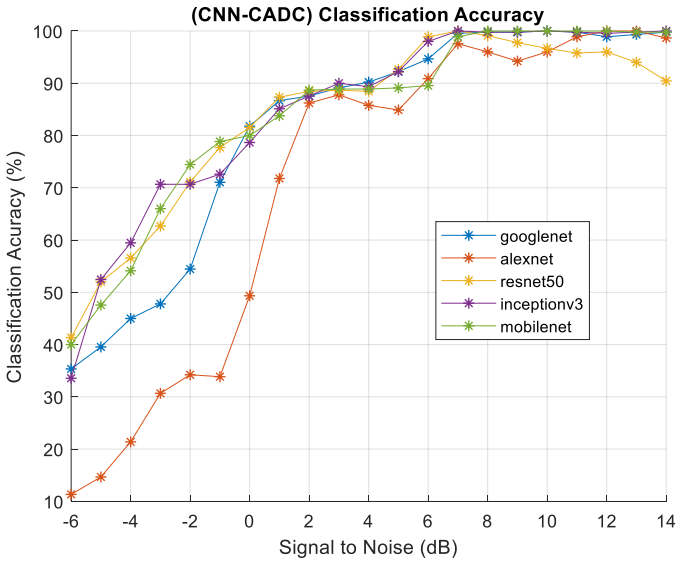


Figure 111. CNN-CADC Baseline Accuracy with Respect to SNR - Known Datasets

Figure 112 shows the overall performance accuracy of the CNN-CADC algorithm. InceptionV3 provides the greatest overall performance at about 85%. Again, for purposes of this study, the networks were not trained to provide optimal performance, but to extract representative features at the last layer of the CNNs for testing the adaptive classifiers.

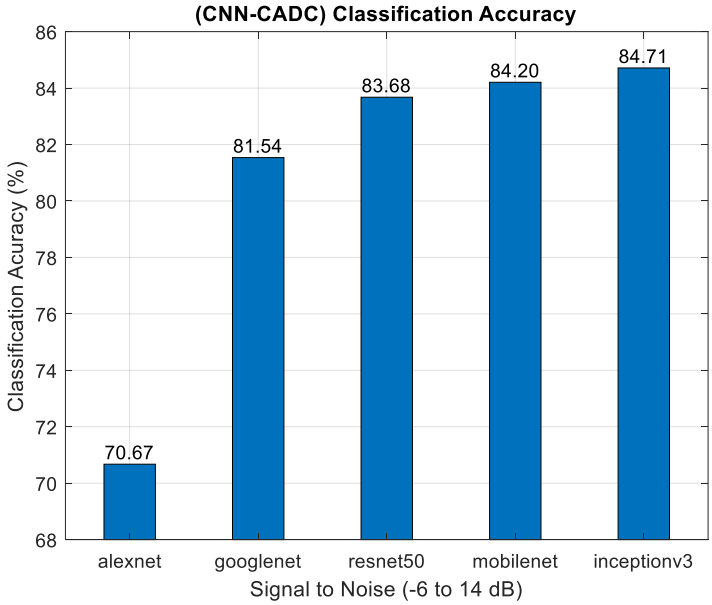


Figure 112. CNN-CADC Overall Accuracy - Known Datasets

6.5.2.1.3 CNN-FHITS

This section provides the performance results of each of the CNN having the last layers replaced with the FHITS algorithm operating in its non-adaptive mode. Figure 113 shows the results of the FHITS networks. Overall, the networks perform well, and overall, as what one might expect with accuracies of all the classifiers increasing as the SNR levels increase. This behavior is

unlike that observed in the CNN and CNN-CADC baseline performances discussed in sections 6.5.2.1.1 and 6.5.2.1.2 respectively.

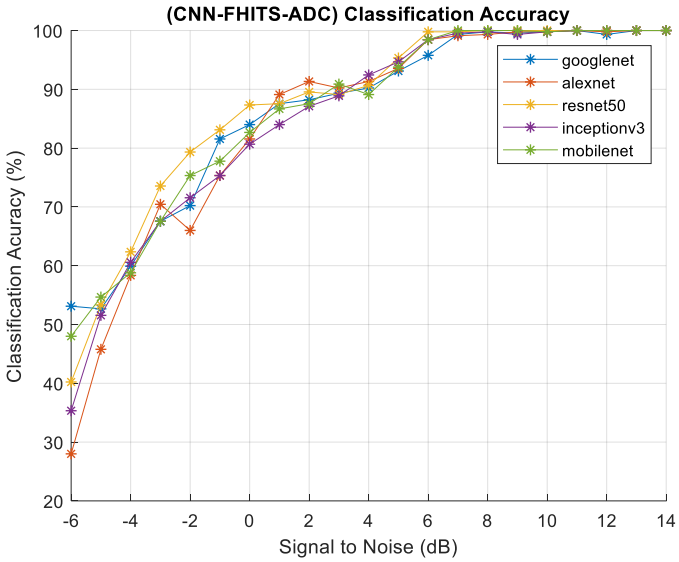


Figure 113. CNN-FHITS Baseline Accuracy with respect to SNR - Known Datasets

Figure 114 shows the overall performance of the CNN-FHITS algorithm across each of the five CNN architectures. ResNet50 provides the greatest overall performance among this group at about 87%. Generally, CNN-FHITS is outperforming CNN-CADC in baseline results.

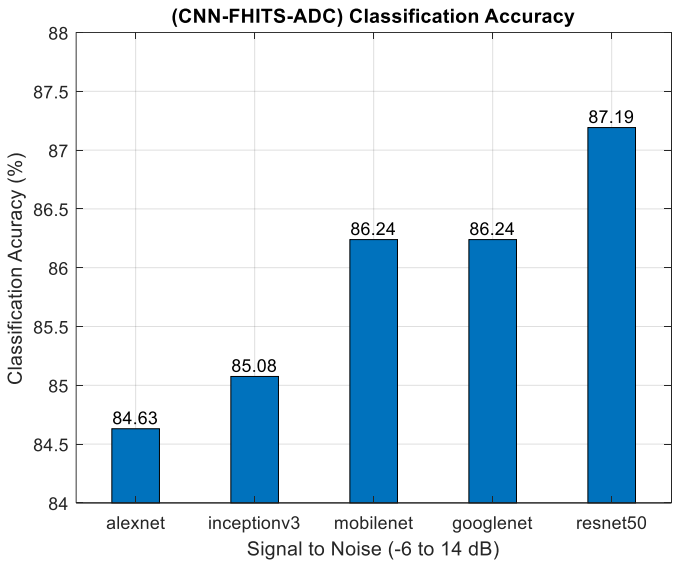


Figure 114. CNN-FHITS Overall Accuracy - Known Datasets

6.5.2.1.4 CNN-DYNG

This section summarizes the results of CNN-DYNG. The hyper-parameters for the DYNG networks are as shown in Table 40. The MAX_NODES parameter was set to a maximum of 100 nodes to limit the growth of the DYNG networks generated. That yields about 11 DYNG nodes representing each of the 9 training classes. The MAX_AGE was set low at 25 to also help minimize the DYNG growth. The desire to minimize the size of the network is primarily driven by limited compute resources available in running the experiment. For training, 7000 '3Channel' images with SNR levels ranging from -6 to 14 dB were used. This provided 450 test samples at each dB level. This was about 100 samples per class at each dB level (there were 9 known classes) from -6, -5, -4, … 14. During training the CNN-DYNG was trained using five (5) epoch cycles. This was done to allow the network to generalize across the training set. Five (5) epochs were arbitrarily chosen to get reasonable performance. For each network, 5 DYNG networks were generated, one for each of the 5 CNNs.

Table 40. DYNG Hyperparameters

Parameter	Value
EB	0.1
EN	0.0006
ALPHA	0.5
D	0.0005
LAMBDA	30
MAX_AGE	25
MAX_NODES	100
NUM_STD	5.0

Figure 115 shows the results of CNN-DYNG. The algorithm performs well, and as one might expect with accuracies of all classifiers increasing as the SNR levels increase. This behavior is unlike that observed in the CNN and CNN-CADC baseline results discussed in sections 6.5.2.1.1 and 6.5.2.1.2 respectively.

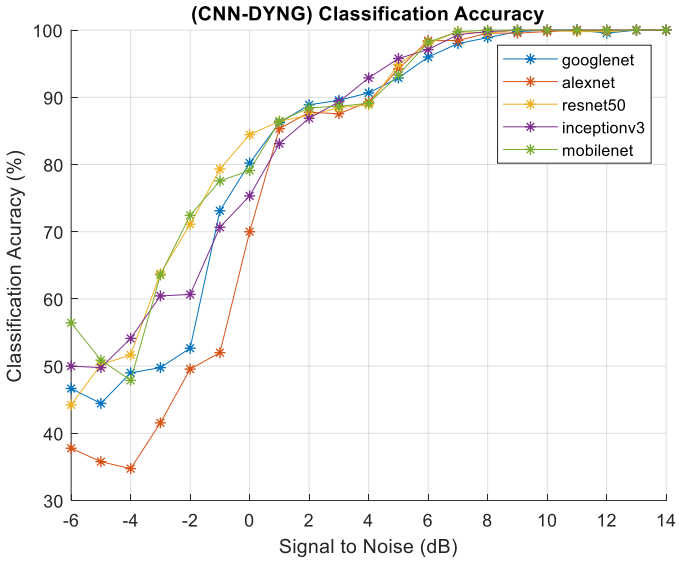


Figure 115. CNN-DYNG Classification Accuracy

Figure 116 shows the overall classification performance of the CNN-DYNG networks, with MobileNet providing the best overall performance of more than 85%.

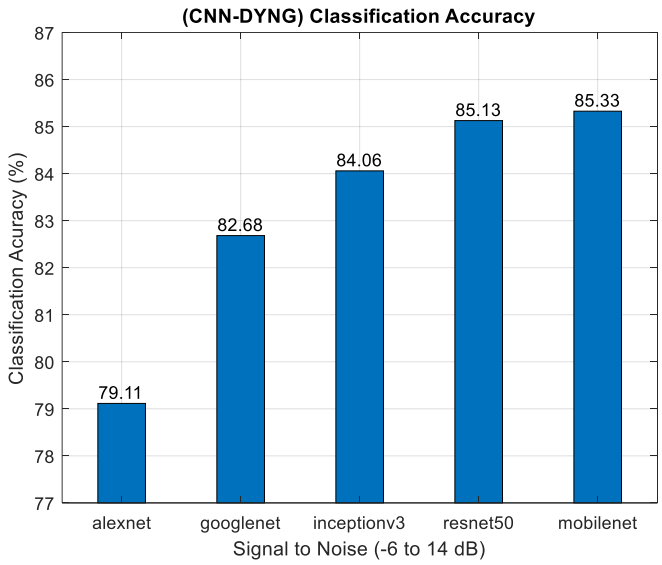


Figure 116. CNN-DYNG Overall Classification Accuracy

6.5.2.1.5 Comparative Summary

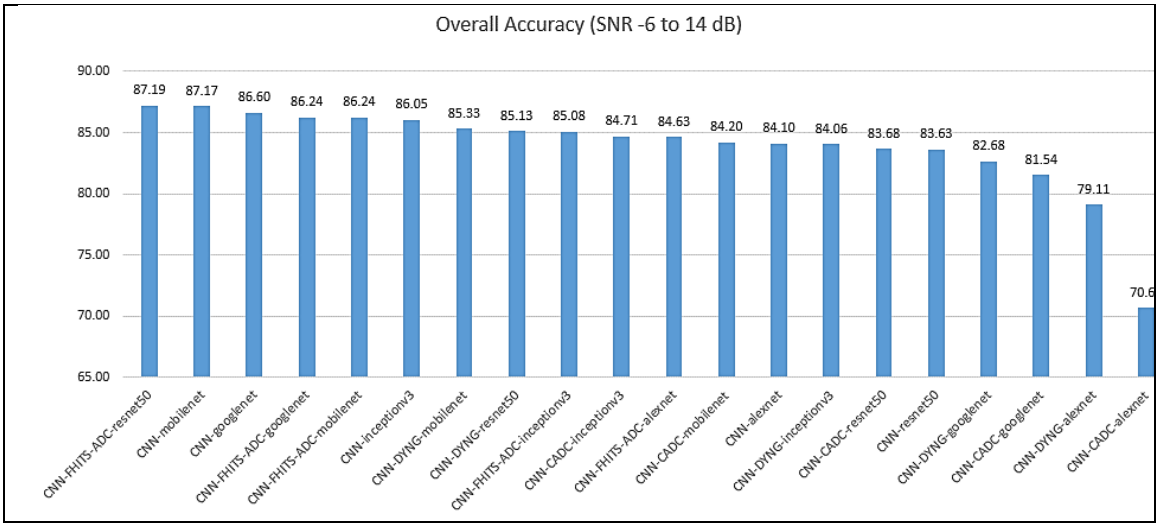


Figure 117. Comparative Summary of Non-Adaptive Accuracies

Figure ***117*** shows a final comparative summary of the baseline accuracies with the most accurate models ordered from left to right. The top three performing algorithms are CNN-FHITS-ResNet50, CNN-MobileNet, and CNN-GoogleNet performing at about 87%. In some cases, the standard softmax layer for the CNNs does not provide the greatest accuracy. For example, CNN-FHITS-ResNet50 provides greater accuracy (87.19%) than CNN-resenet50 (83.63%). Another similar case is with CNN-FHITS-AlexNet. It is possible that further training on the CNNs could lead to resolving these discrepancies. Nevertheless, most of the configurations provide reasonable baselines to use them as starting points for the adaptive algorithms discussed in the following section.

6.5.2.2 Adaptive Performance

As a reminder the overall design goals of these series of tests were to take advantage of using the five CNNs as feature extractors for the RF waveform data, and to use the feature vectors as inputs to test the four (4) adaptive classification algorithms CADC, FHITS, DYNG-FHITS, and DYNG-CADC. For each of these classifiers an overall accuracy discussion is provided and then followed by a more detailed discussion.

As a brief reminder of the test methodology, the classifiers are randomly exposed to zero (0) through six (6) anomaly classes and nine (9) known classes. Performance is assessed with respect to two SNR ranges, the noisier -6 to 14 dB range, and the cleaner 4 to 14 dB range. Tests

are carried out by either freezing or adapting the sigma parameters of the anomaly nodes as inputs are assigned to the corresponding anomaly class. When sigma is not frozen and the input stimuli are classified as anomalies, the sigma value of the selected anomaly class will adapt. For all anomaly classes, the centroid vectors always adapt as input stimuli are assigned to a specific anomaly class. Finally, the known centroid parameters derived from training on the known data sets are not permitted to change once online.

6.5.2.2.1 CNN-CADC

This section presents the results of how the adaptive CADC algorithm performs when presented with known and anomalous data. The first sub-section presents a comparative summary of the overall accuracies of the CADC algorithm applied among the five CNNs, and the second sub-section presents a more detailed discussion on the performance results.

6.5.2.2.1.1 Overall Accuracy

This section shows the overall accuracy of the CADC algorithm with respect to the number of anomalies input to the adaptive classifiers via the CNNs. In Figure 118, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with σ-adapt with lower and higher SNR levels respectively. Overall, a significant result shown in the figures is that the overall accuracy of the classifiers is maintained as anomalies are introduced.

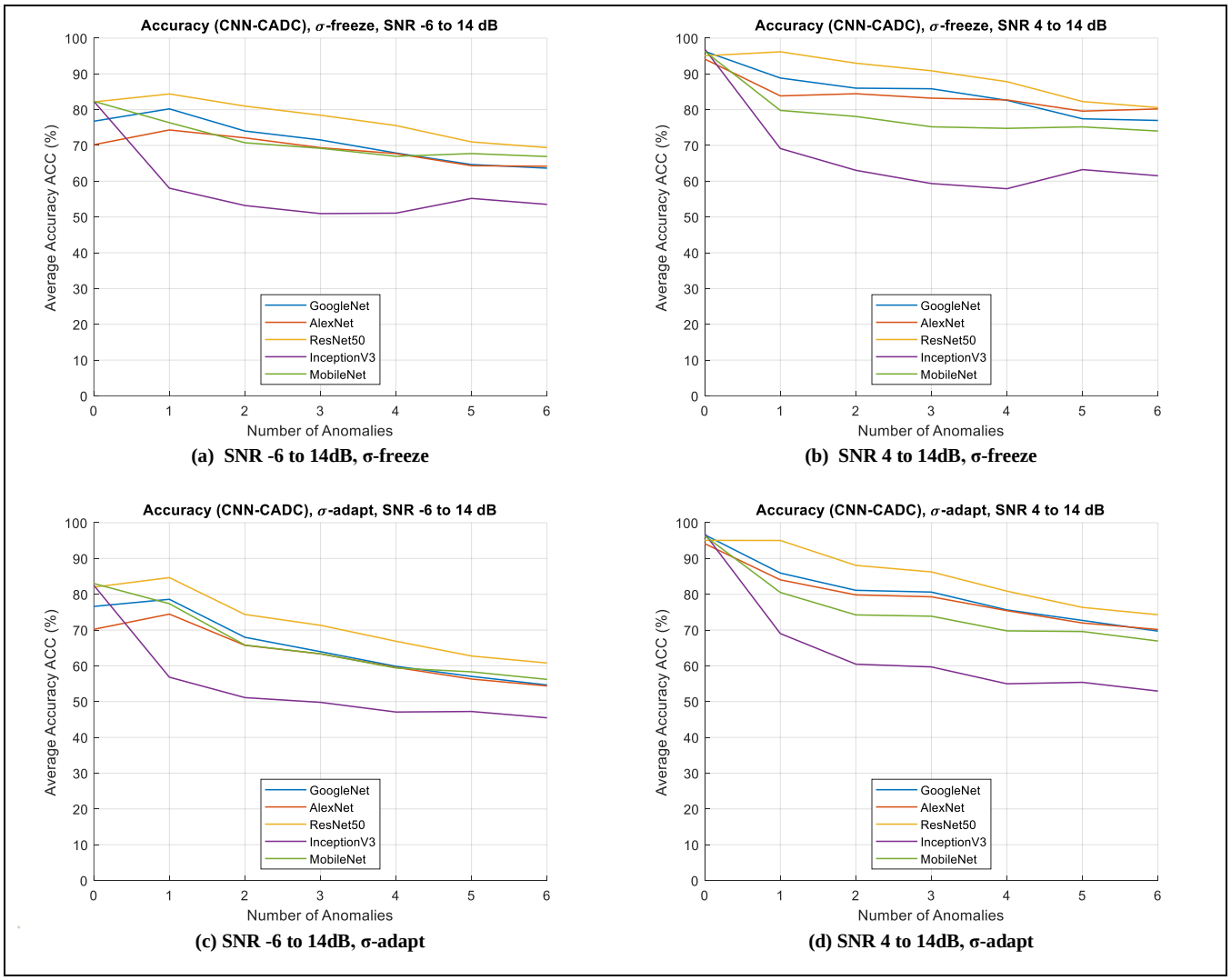


Figure 118. CNN-CADC Overall Accuracy (ACC)

As expected, performance is greater as SNR levels span the highest levels, however unexpectedly, σ-frozen cases tend to perform better than the σ-adapt cases. ResNet50 gives the best overall performance, as indicated by the orange lines in all sub-figures rising above all other legend colors. InceptionV3 is the poorest performers overall as indicated in the green line plots. For all CNNs, as the number of anomalies increases, adaptability allows the algorithms to sustain a greater level of performance than if they did not adapt. A more detailed discussion follows in the next subsection by analyzing the TNR, ACC, TPR, and NAPR performance metrics.

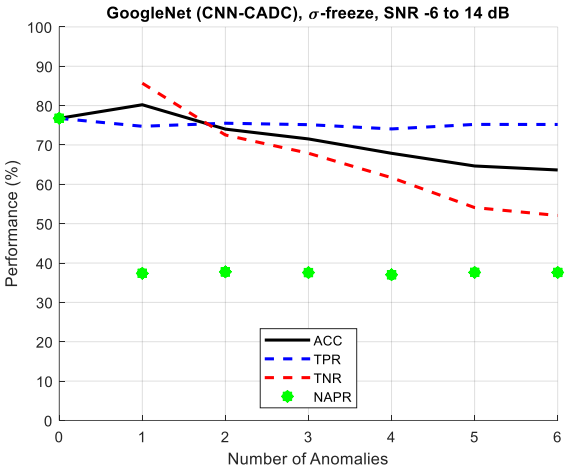
6.5.2.2.1.2 Detailed Results

This section summarizes the more detailed results of the DYNG-FHITS algorithm applied to the different CNNs studied. In Figure 119 through Figure 123, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with *σ*-adapt with lower and higher SNR levels respectively.

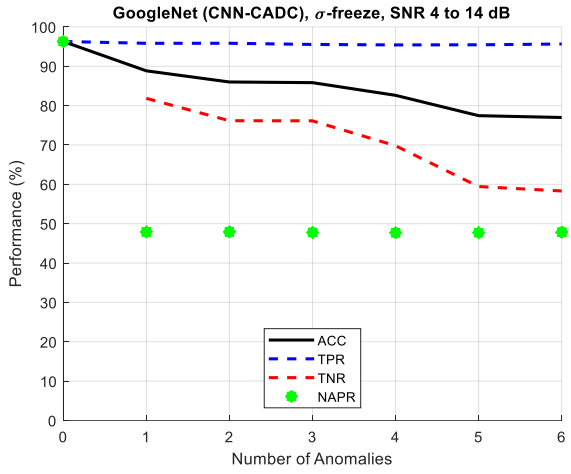
In all cases, the overall accuracy (ACC) of the higher SNR levels shows greater performance than the lower SNR levels. Promising results show that the accuracy of detecting the known waveforms (TPR) has minimal degradation as the number of anomalies increases, and good overall performance (ACC) is maintained well above the non-adaptive positive rate (NAPR).

The accuracy of detecting anomalies (TNR) shows performance of the anomaly detection and clustering ability, and in all cases as the number of anomalies increases, there is a tendency for TNR to degrade.

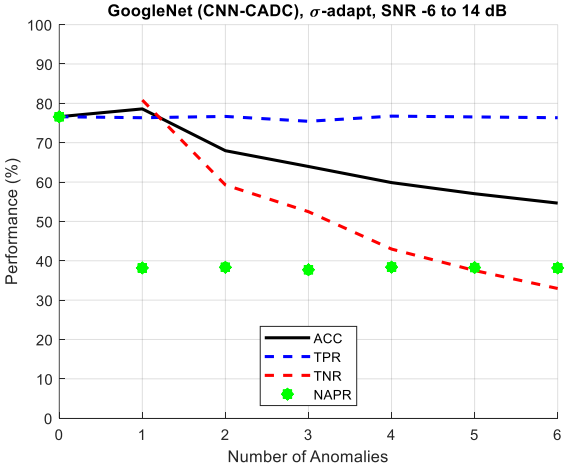
With respect to σ-frozen or adapting, in general one can see that keeping σ-frozen tends to provide the best and most stable performance overall. This is an unexpected outcome, because σ-adapt is expected to capture the true variability of the anomalous data as the classifiers learn from the samples, and therefore provide greater performance.



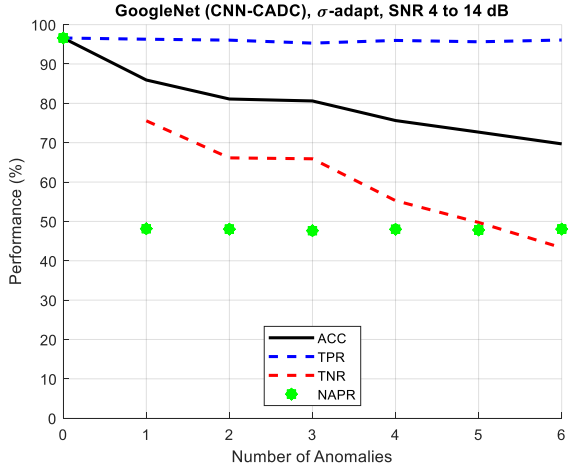
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

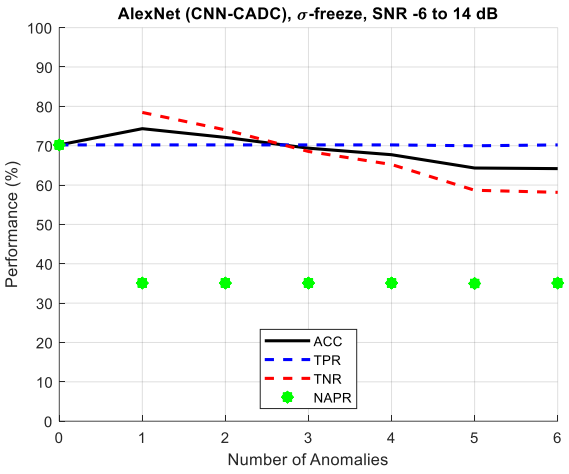


(c) SNR -6 to 14dB, σ-adapt

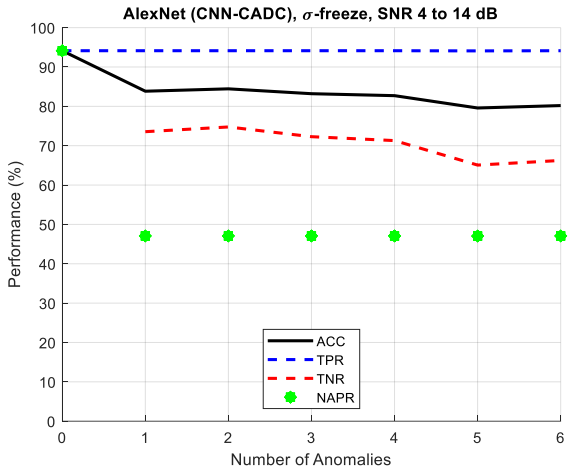


(d) SNR 4 to 14dB, σ-adapt

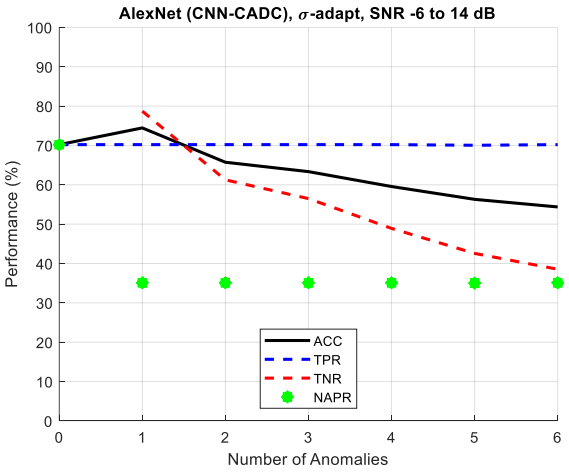
Figure 119. CADC GoogleNet Performance



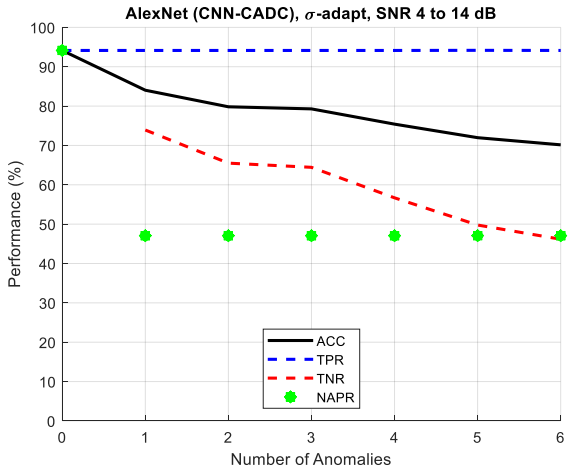
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

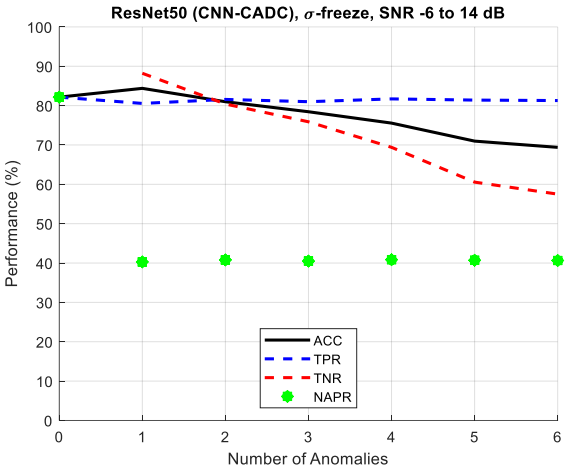


(c) SNR -6 to 14dB, σ-adapt

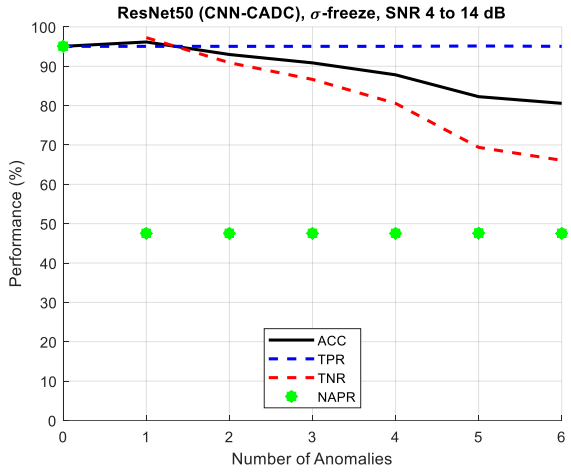


(d) SNR 4 to 14dB, σ-adapt

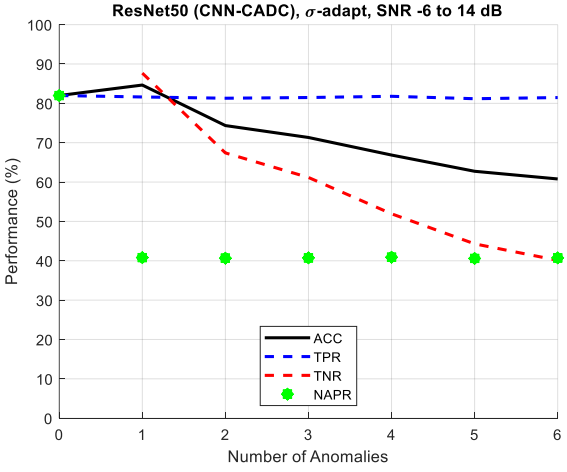
Figure 120. CADC AlexNet Performance



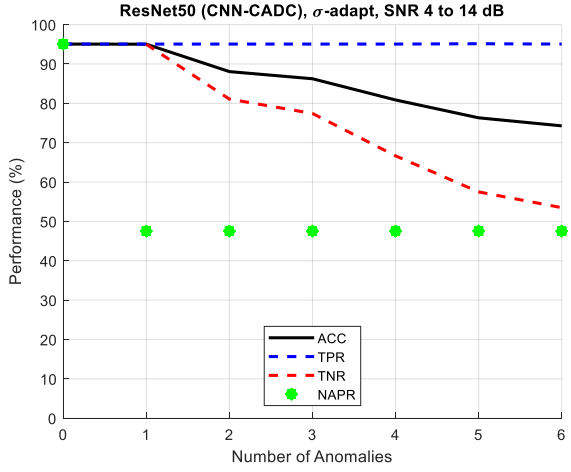
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

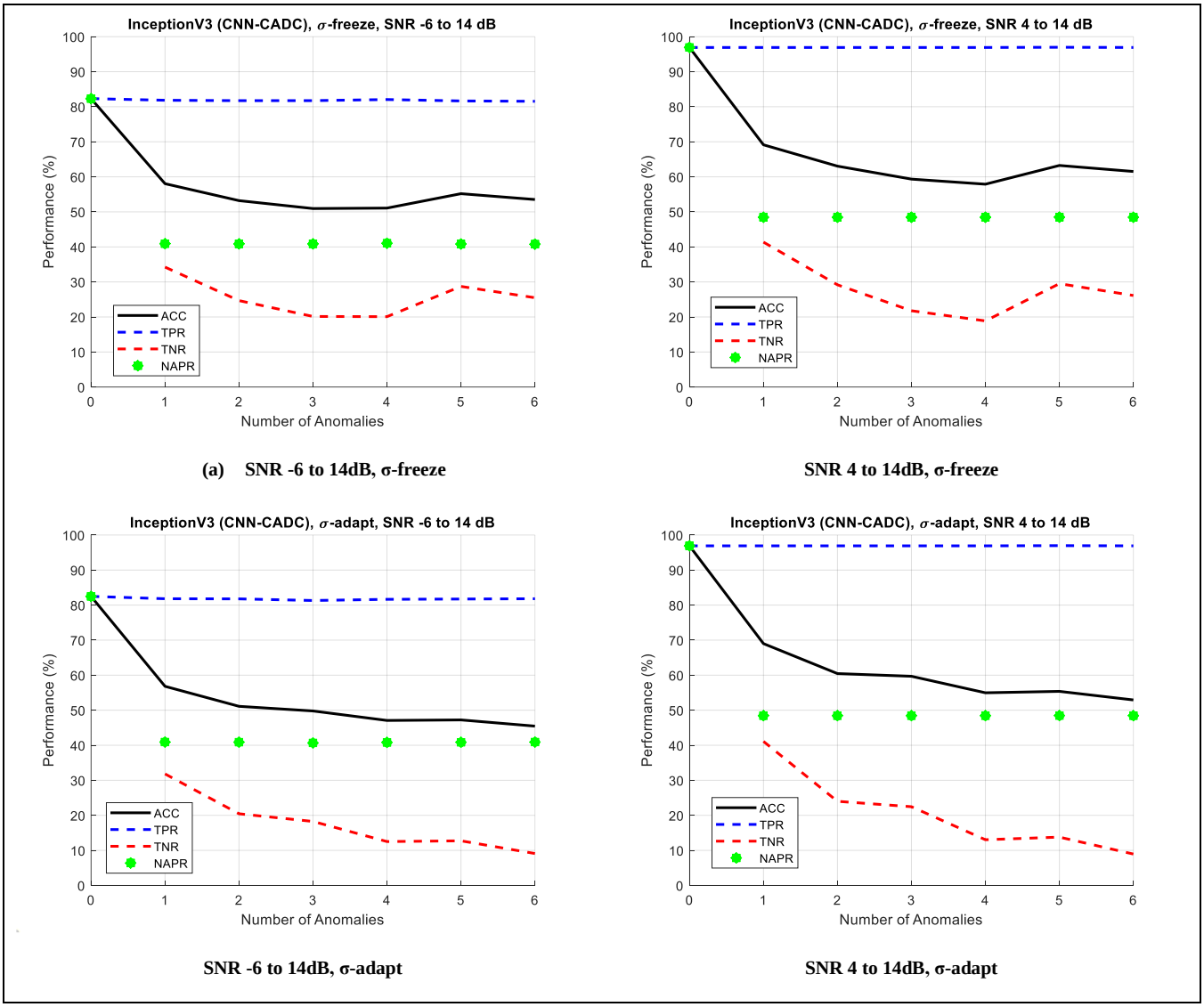


(c) SNR -6 to 14dB, σ-adapt



(d) SNR 4 to 14dB, σ-adapt

Figure 121. CADC ResNet50 Performance



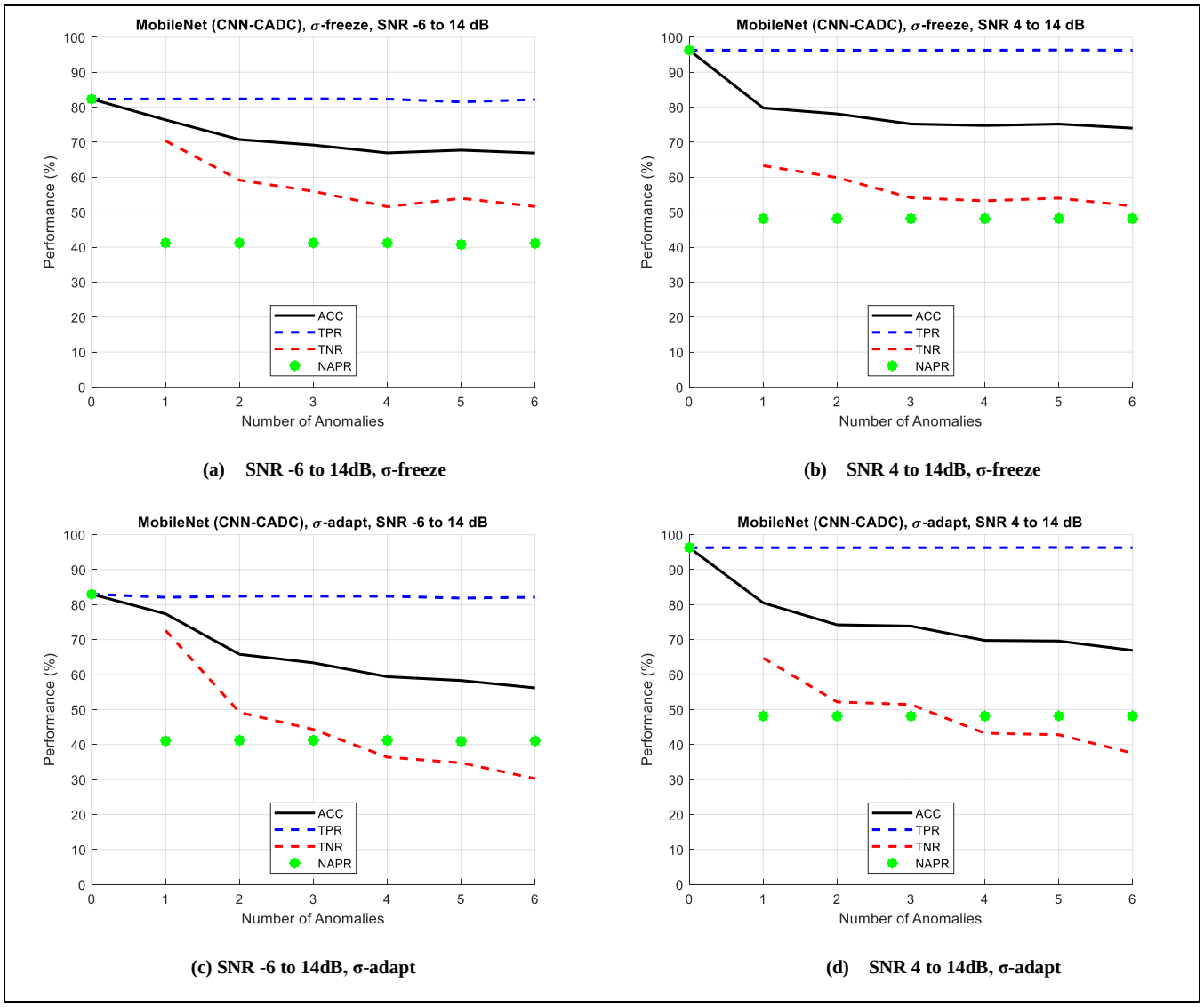
(a) SNR -6 to 14dB, σ-freeze

SNR 4 to 14dB, σ-freeze

SNR -6 to 14dB, σ-adapt

SNR 4 to 14dB, σ-adapt

Figure 122. CADC InceptionV3 Performance



(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

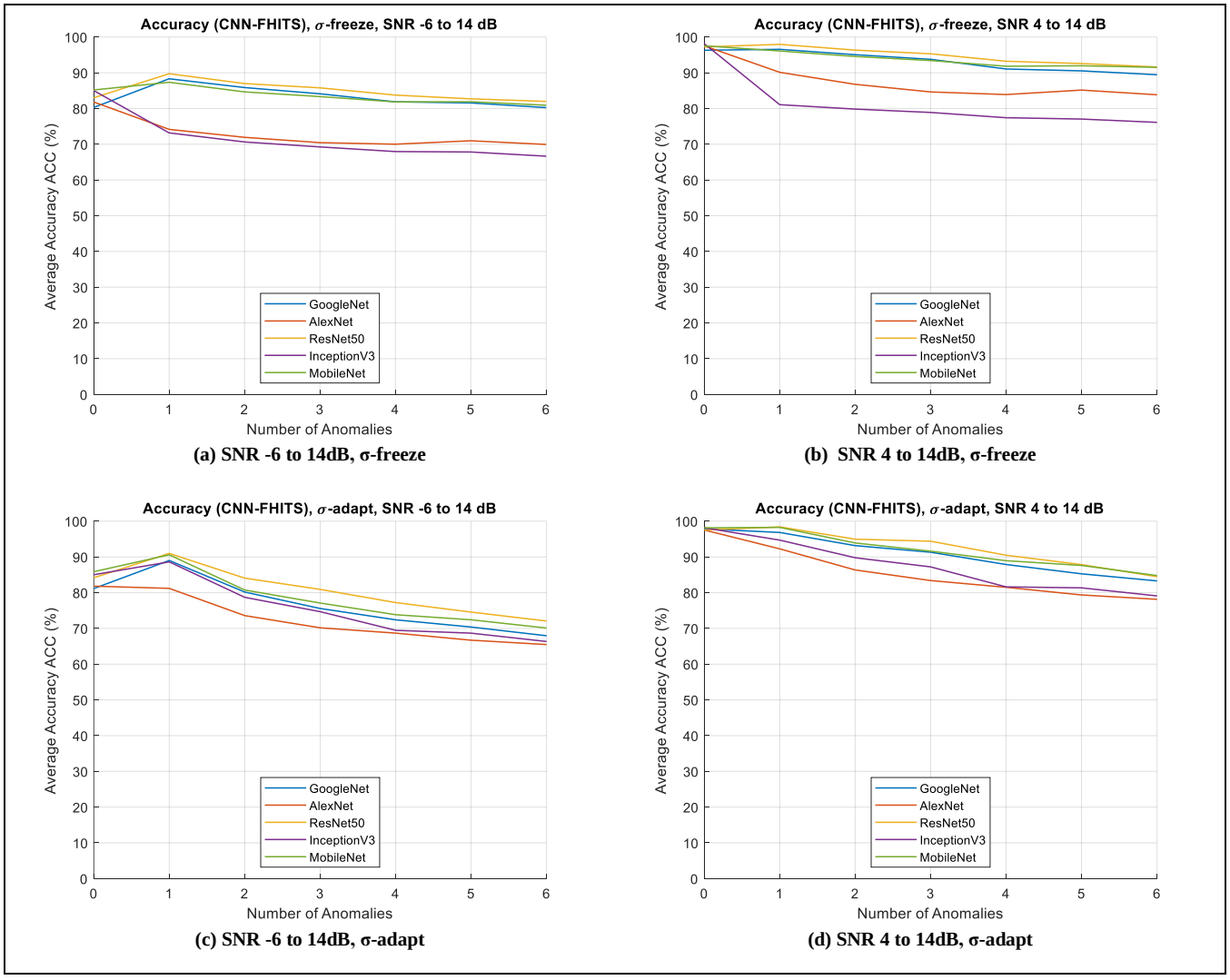
Figure 123. CADC MobileNet Performance

6.5.2.2.2 CNN-FHITS

This section presents the results of how the adaptive FHITS algorithm performs when presented with known and anomalous data. The first sub-section presents a comparative summary of the overall accuracies of the algorithm applied among the five CNNs, and the second sub-section presents a more detailed discussion of the performance results.

6.5.2.2.2.1 Overall Accuracy

This section shows the overall accuracy of the FHITS algorithm with respect to the number of anomalies input to the adaptive classifiers via the CNNs. In Figure 124, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with σ-adapt with lower and higher SNR levels respectively.



(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

Figure 124. CNN-FHITS Overall Accuracy (ACC)

As expected, performance is greater as the SNR levels span the highest levels, however unexpectedly, σ-frozen cases tend to perform better than the σ-adapt cases. ResNet50 gives the best overall performance, as indicated by the orange lines in all sub-figures rising above all other legend colors. AlexNet and InceptionV3 are the poorest performers overall. For all CNNs, as the number of anomalies increases, adaptability allows the algorithms to sustain a greater level of performance than if they did not adapt. A more detailed discussion follows in the next subsection by analyzing the TNR, ACC, TPR, and NAPR performance metrics.

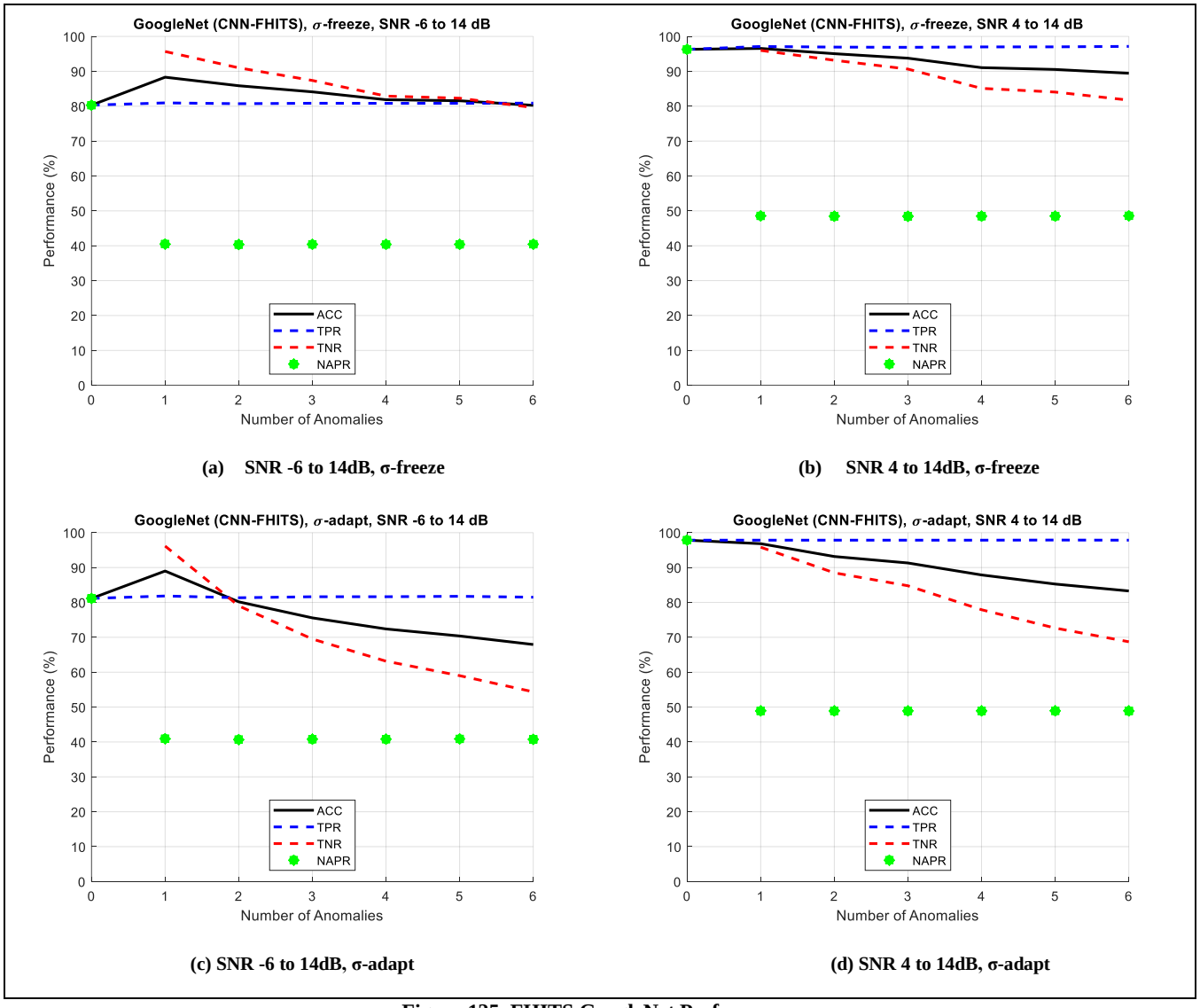
6.5.2.2.2.2 Detailed Results

This section provides more detailed results of the FHITS algorithm applied to the different CNNs studied. In Figure 125 through Figure 129, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with *σ*-adapt with lower and higher SNR levels respectively. Overall, FHITS provides acceptable adaptive performance because ACC remains well above NAPR as anomalies are introduced.

In all cases the overall accuracy (ACC) of the higher SNR levels shows greater performance than lower SNRs. The accuracy of detecting the known waveforms (TPR) has some degradation as the number of anomalies increases, however in general, performance is maintained above the non-adaptive performance (NAPR).

An unexpected result is that σ-adapt performs more poorly than σ-freeze, and this is unexpected because σ-adapt is presumed to capture the true variance of the anomaly classes as they stimulate the algorithm. This will require further investigation.

The accuracy of detecting anomalies, true negative rate (TNR), in all cases shows that as the number of anomalies increases there is a tendency for TNR to degrade, and this causes the overall accuracy (ACC) to degrade as well.



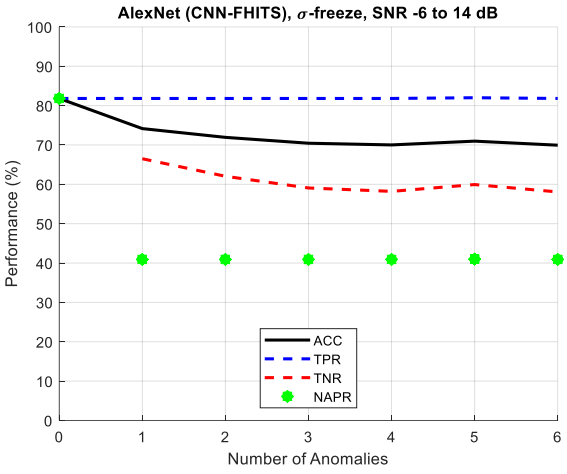
(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

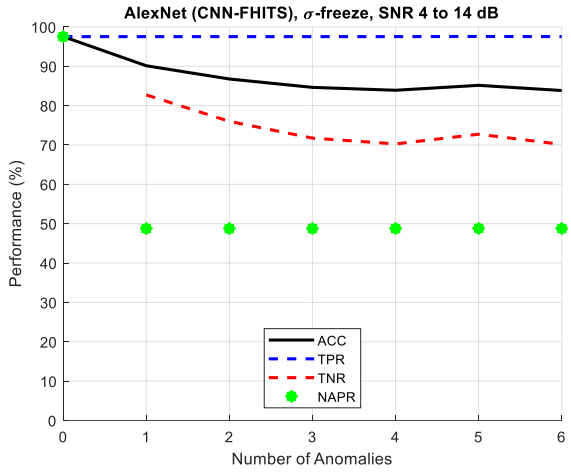
(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

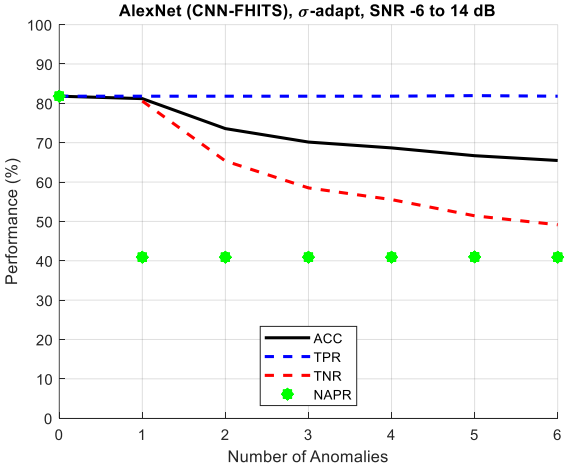
Figure 125. FHITS GoogleNet Performance



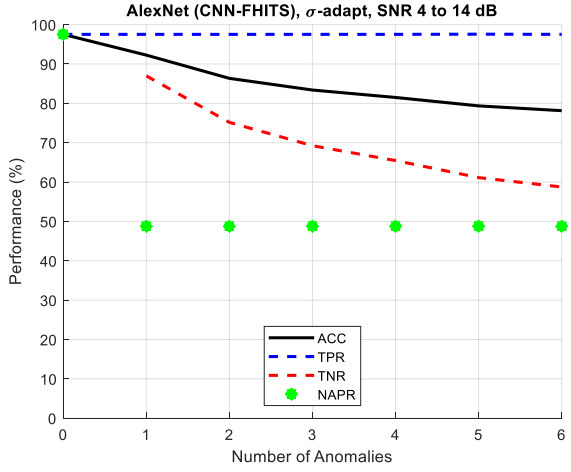
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

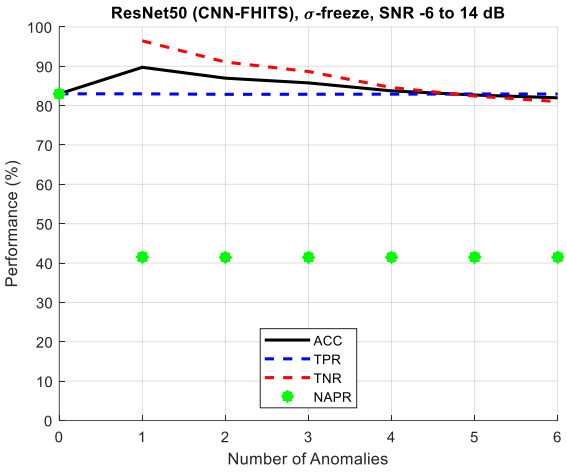


(c) SNR -6 to 14dB, σ-adapt

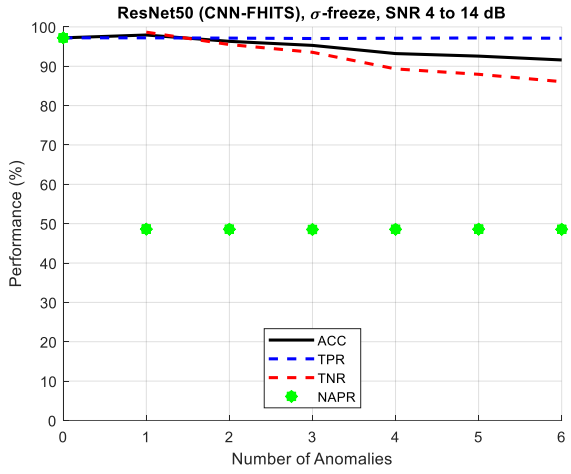


(d) SNR 4 to 14dB, σ-adapt

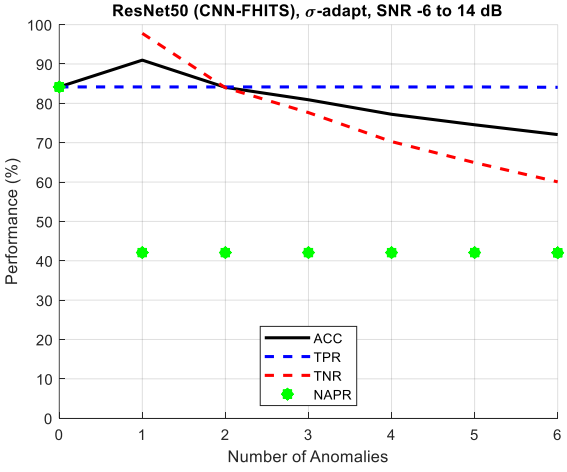
Figure 126. FHITS AlexNet Performance



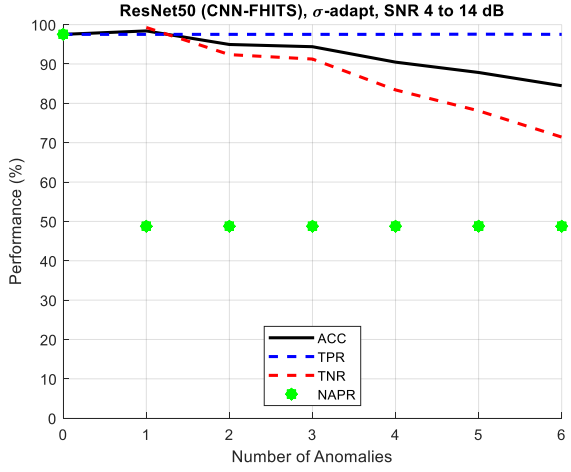
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

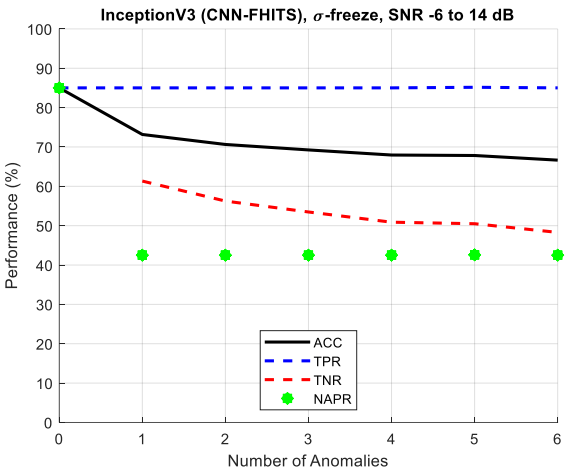


(c) SNR -6 to 14dB, σ-adapt

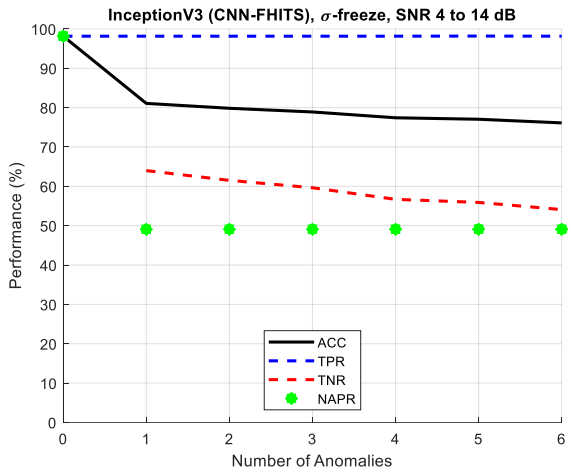


(d) SNR 4 to 14dB, σ-adapt

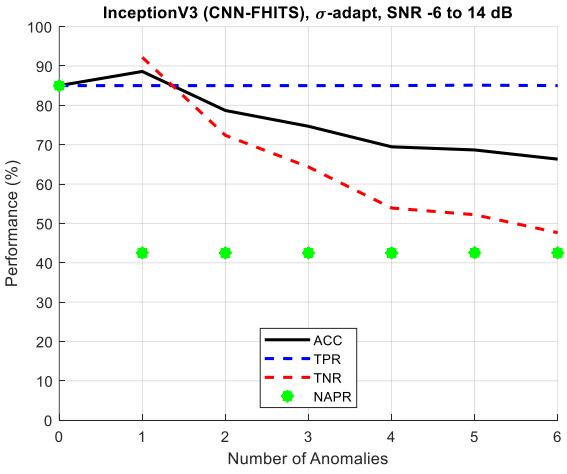
Figure 127. FHITS ResNet50 Performance



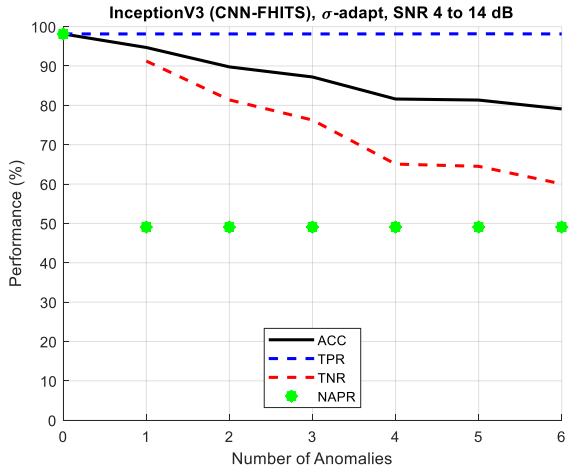
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

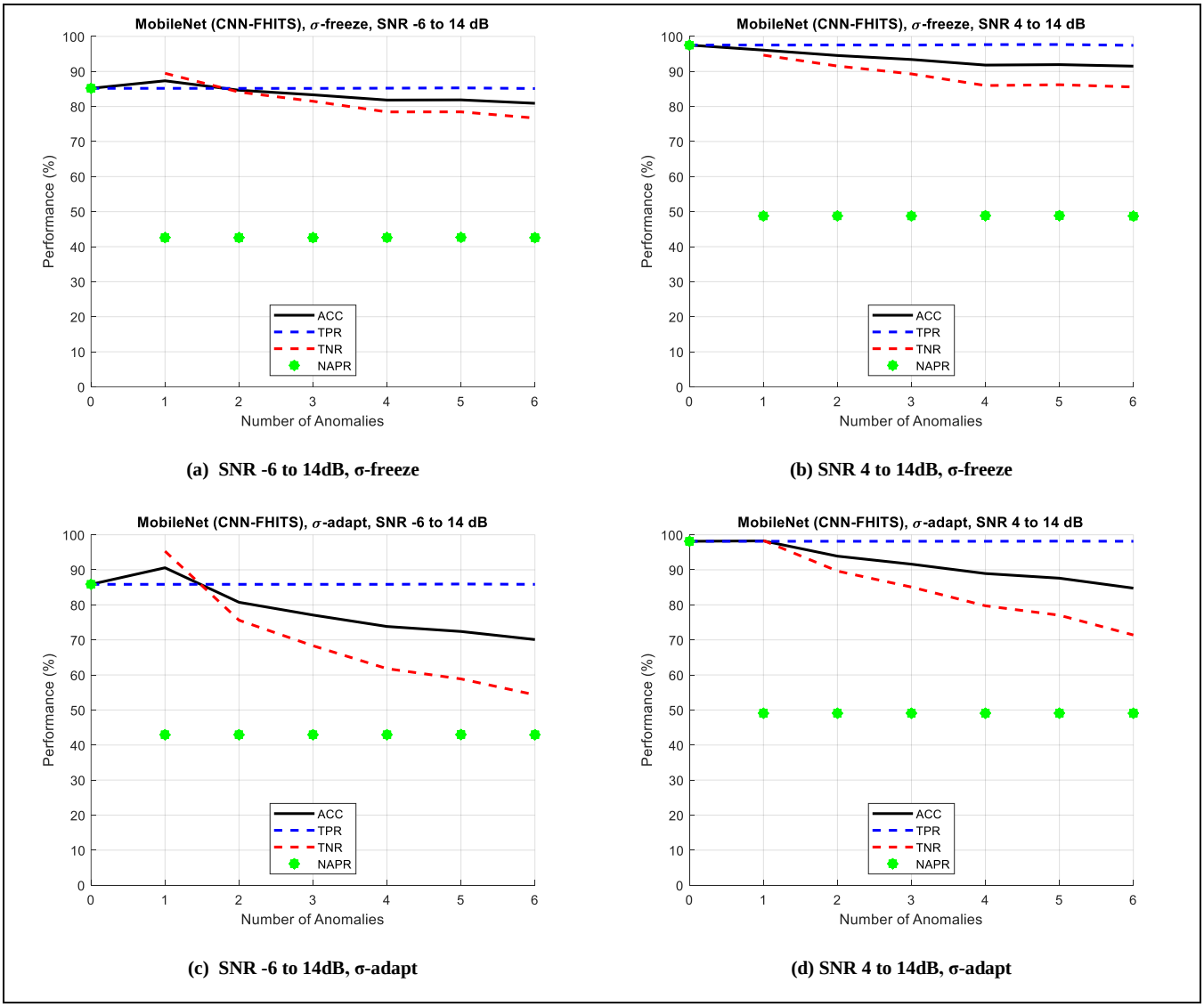


(c) SNR -6 to 14dB, σ-adapt



(d) SNR 4 to 14dB, σ-adapt

Figure 128. FHITS InceptionV3 Performance



(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

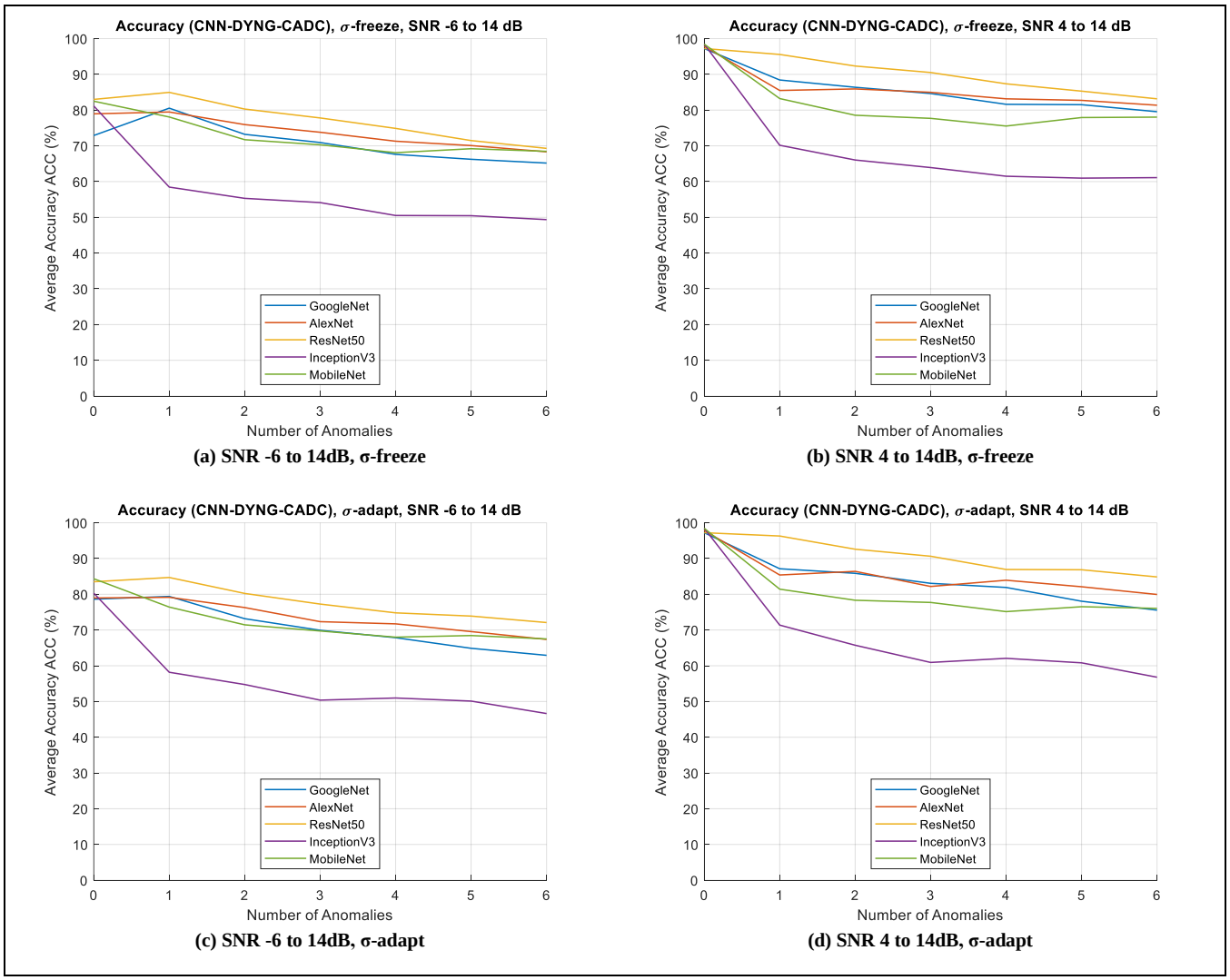
Figure 129. FHITS MobileNet Performance

6.5.2.2.3 CNN-DYNG-CADC

This section presents the results of how the adaptive DYNG-CADC algorithm performs when presented with known and anomalous data. The first sub-section herein presents a comparative summary of the overall accuracies of the DYNG-CADC algorithm applied among the five CNNs, the second sub-section presents a more detailed discussion on the performance results.

6.5.2.2.3.1 Overall Accuracy

This section shows the overall accuracy of the DYNG-CADC algorithm with respect to the number of anomalies input to the adaptive classifiers via the CNNs. In Figure 130, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with σ-adapt with lower and higher SNR levels respectively.



(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

Figure 130. CNN-DYNG-CADC Overall Accuracy (ACC)

As expected, performance improvement is more apparent as the SNR levels span the highest levels. As unexpected, σ-frozen cases tend to perform better than the σ-adapt cases. ResNet50 gives the best overall performance, as indicated by the orange lines in all sub-figures rising above all other legend colors, whereas InceptiionV3 gives poor performance with rapidly degradation as anomalies are introduced.

For all CNNs except InceptionV3, as the number of anomalies increases, the adaptability of the classification algorithms allows them to sustain a greater level of performance than if they did not adapt. A more detailed discussion follows in the next subsection by analyzing the TNR, ACC, TPR, and NAPR performance metrics.

6.5.2.2.3.2 Detailed Results

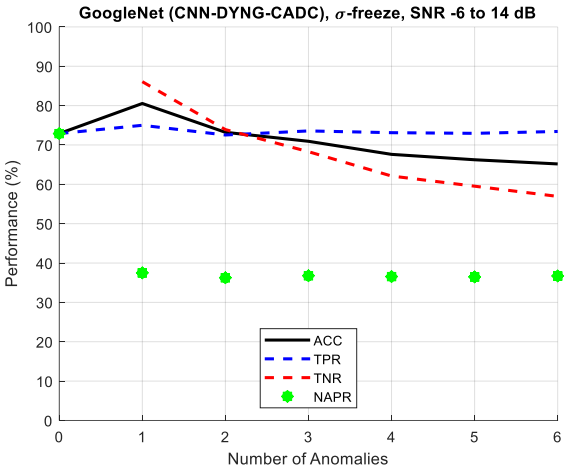
This section provides more detailed results of the DYNG-CADC algorithm applied to the different CNNs studied. In Figure 131 through Figure 135, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with σ-adapt with lower and higher SNR levels respectively.

In all cases the overall accuracy (ACC) of the higher SNR levels shows greater performance than lower SNR levels. The accuracy of detecting the known waveforms (TPR) has some degradation as the number of anomalies increase, however in general, performance is maintained above the non-adaptive performance (NAPR), indicating that the adaptive process is adding significant value in maintaining performance as anomalies are introduced.

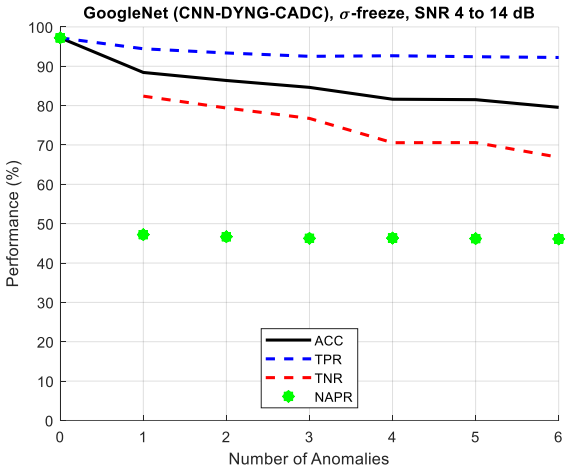
An unexpected result is that the σ-adapt performs more poorly than σ-freeze, however it is expected that σ-adapt should be able to capture the true variance of the anomaly classes as they stimulate the algorithm.

The accuracy of detecting anomalies, true negative rate (TNR), in all cases shows that as the number of anomalies increases there is a tendency for TNR to degrade, and therefore causes the overall accuracy ACC to degrade as well.

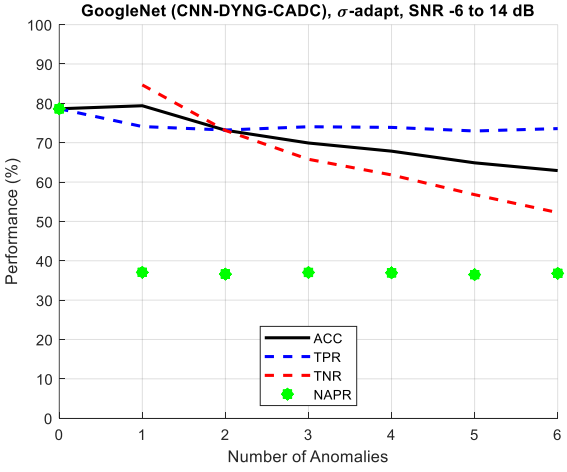
A particularly poor case is with InceptionV3 is shown in Figure 134, where the anomaly detection TNR values start at about 38-45% accuracies with only one anomaly input, and then degrades to about 12-25% as more anomalies are introduced. The cause of this behavior requires further investigation and may suggest the need to adjust the CADC hyperparameters to obtain better performance.



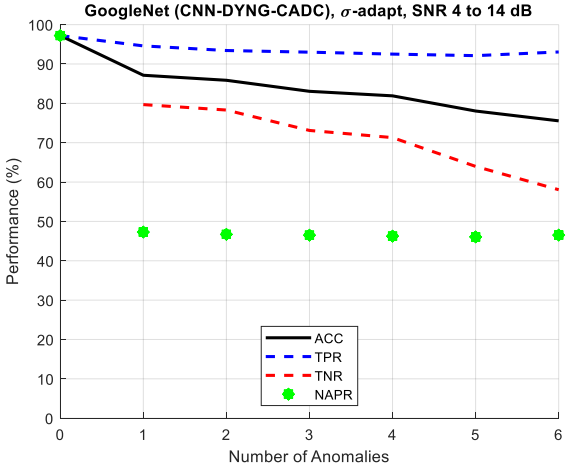
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

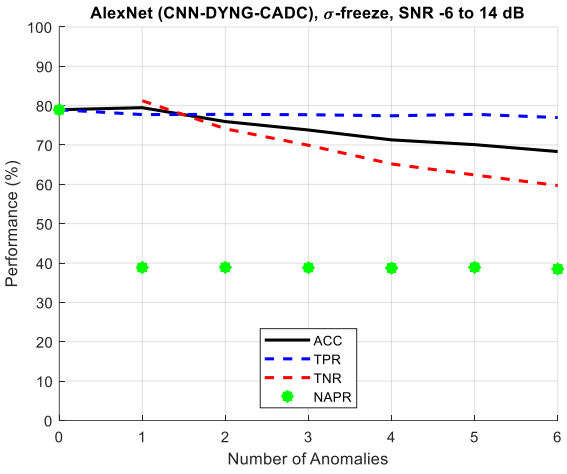


(c) SNR -6 to 14dB, σ-adapt

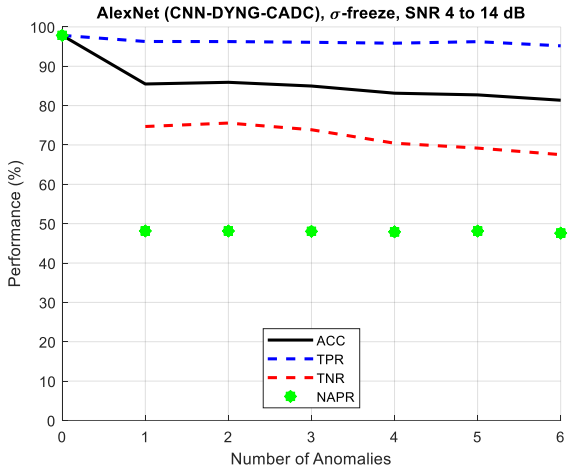


(d) SNR 4 to 14dB, σ-adapt

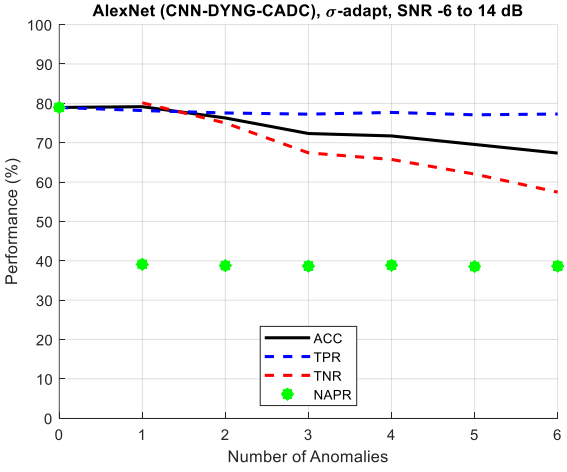
Figure 131. DYNG-CADC GoogleNet Performance



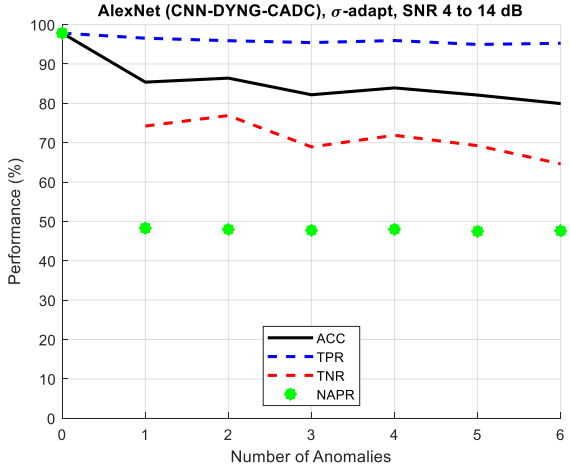
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

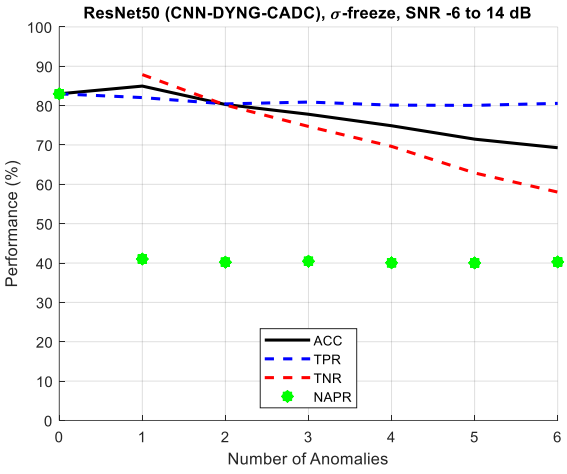


(c) SNR -6 to 14dB, σ-adapt

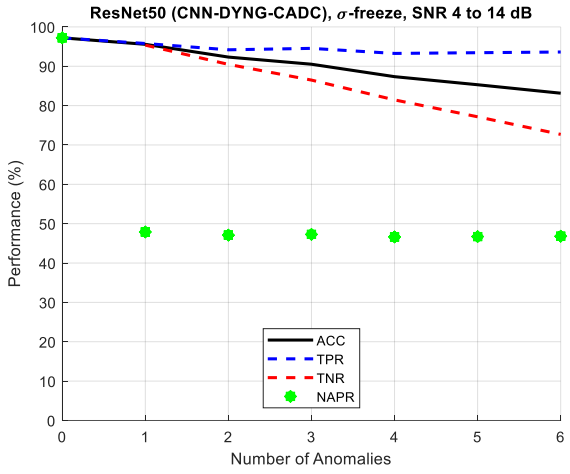


(d) SNR 4 to 14dB, σ-adapt

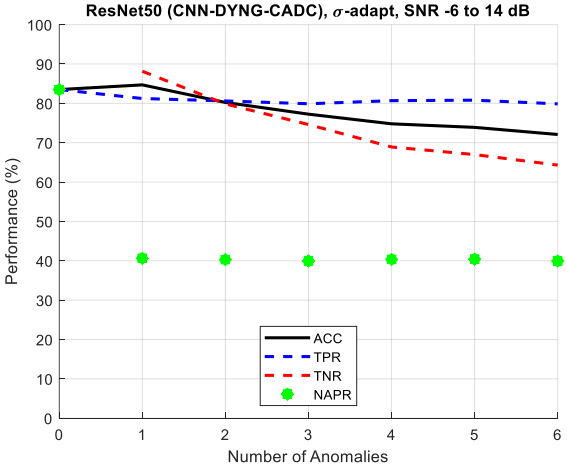
Figure 132. DYNG-CADC AlexNet Performance



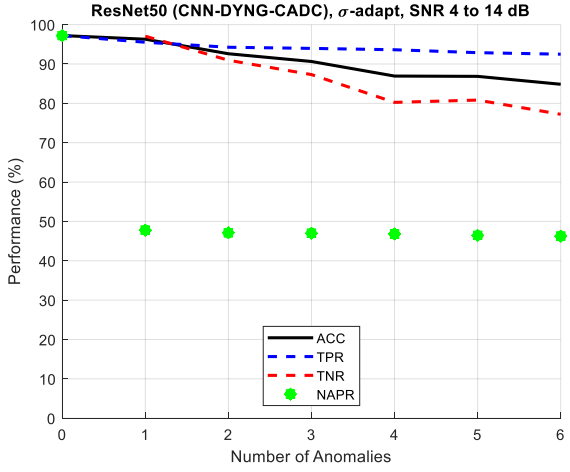
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

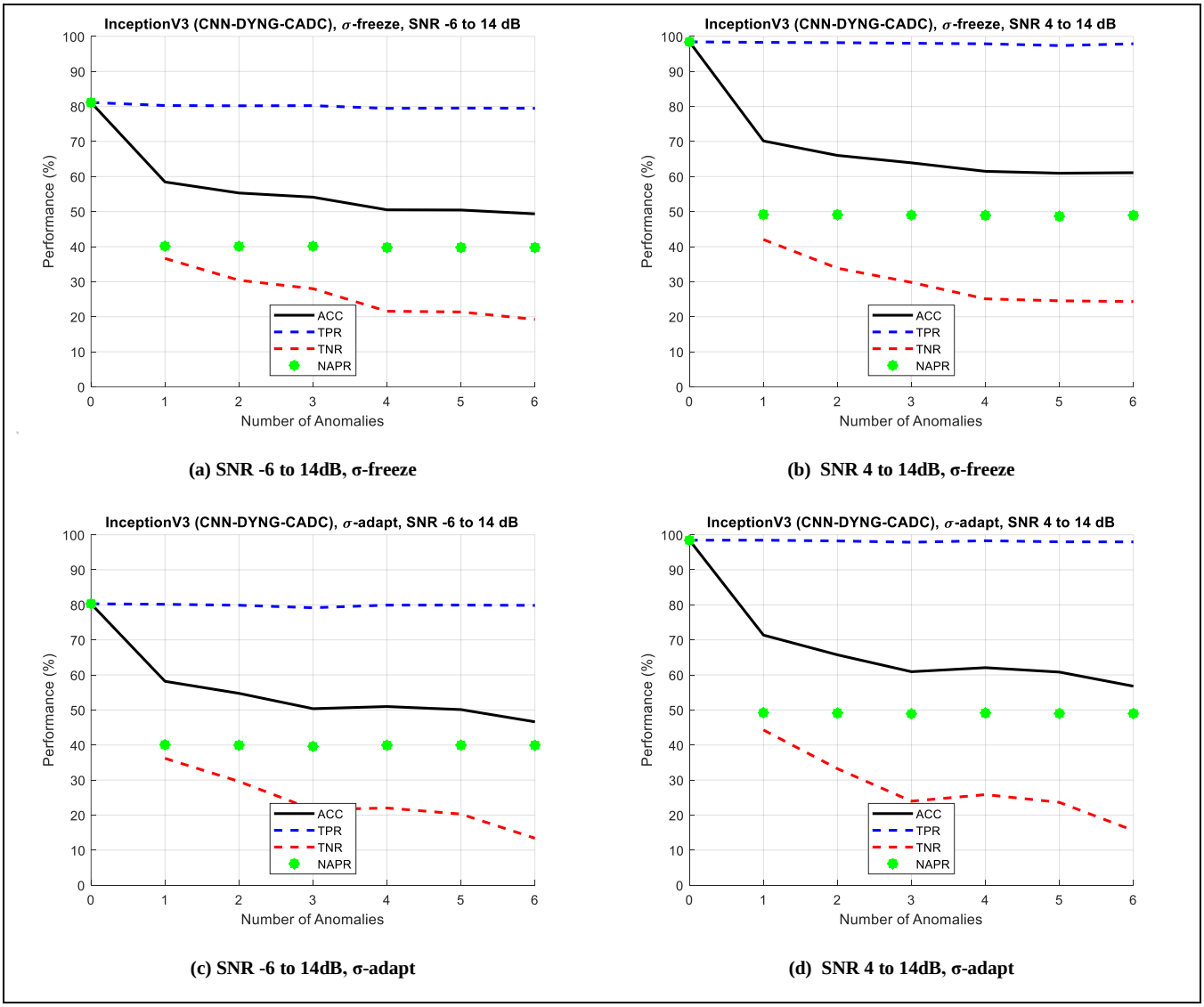


(c) SNR -6 to 14dB, σ-adapt



(d) SNR 4 to 14dB, σ-adapt

Figure 133. DYNG-CADC ResNet50 Performance



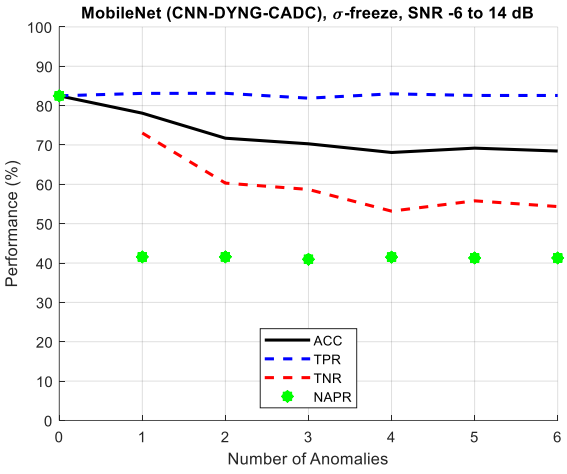
(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

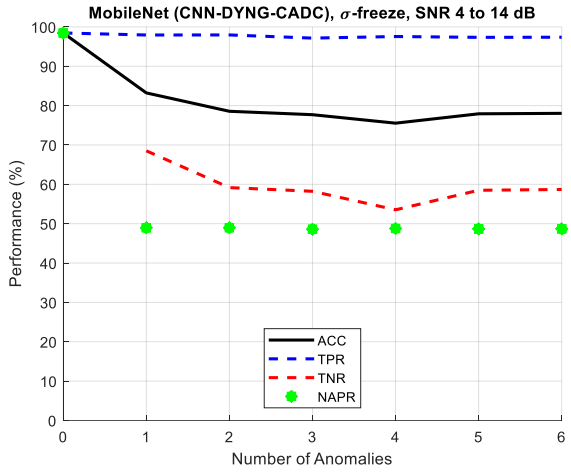
(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

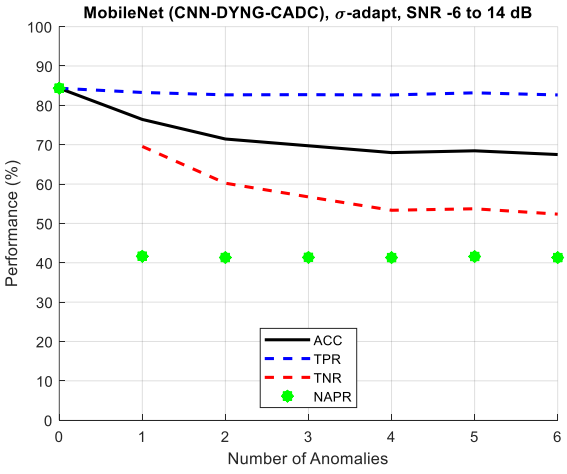
Figure 134. DYNG-CADC InceptionV3 Performance



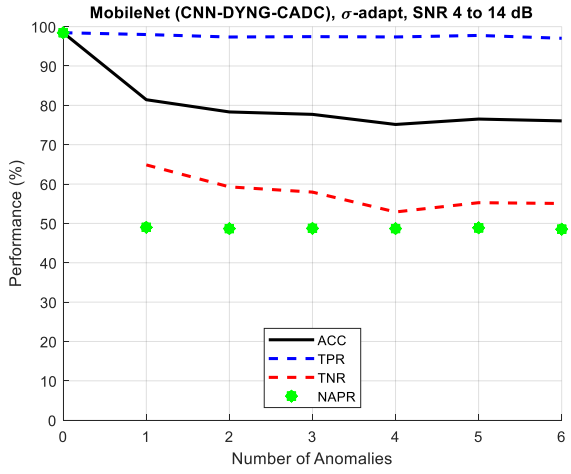
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze



(c) SNR -6 to 14dB, σ-adapt



(d) SNR 4 to 14dB, σ-adapt

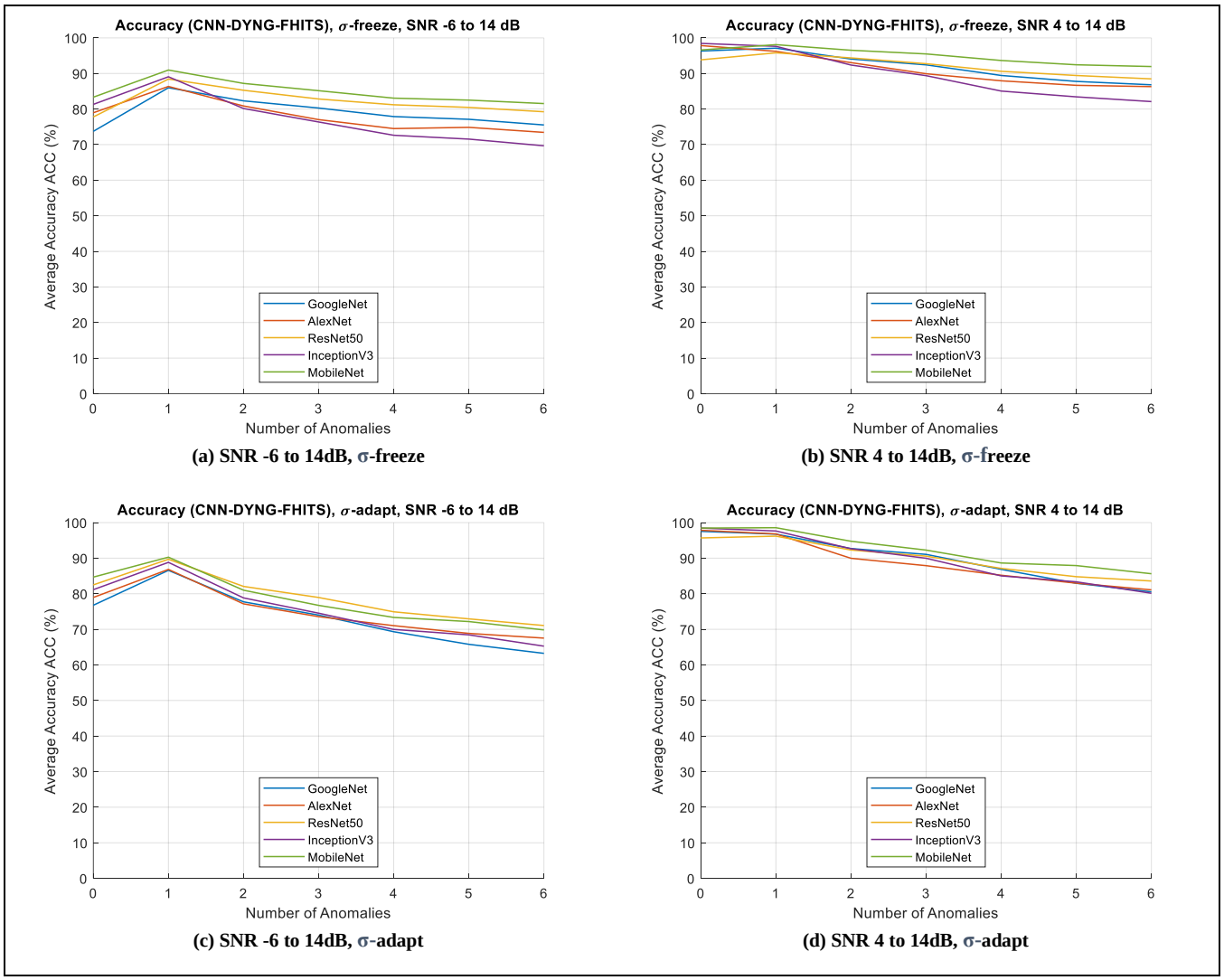
Figure 135. DYNG-CADC MobileNet Performance

6.5.2.2.4 CNN-DYNG-FHITS

This section presents the results of how the adaptive DYNG-FHITS algorithm performs when presented with known and anomalous data. The first sub-section herein presents a comparative summary of the overall accuracies of the DYNG-FHITS algorithm applied among the five CNNs, the second sub-section presents a more detailed discussion on the performance results.

6.5.2.2.4.1 Overall Accuracy

This section shows the overall accuracy of the DYNG-FHITS algorithm with respect to the number of anomalies input to the adaptive classifiers via the CNNs. In Figure 136, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively, while sub-figures (c) and (d) show results with σ-adapt with lower and higher SNR levels respectively.



(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

Figure 136. CNN-DYNG-FHITS Overall Accuracy

As expected, performance improvement is more apparent as the SNR levels span the highest levels. As unexpected the σ-frozen cases tend to perform better than the σ-adapt cases. MobileNet appears to give the best overall performance, as noted by the green lines in sub-figures (a), (b), and (d) generally rising above all other legend colors.

Overall, as the number of anomalies increases the adaptability of the classification algorithms allow them to sustain a greater level of performance than if they did not adapt, and this

claim is substantiated by discussions in next subsection by analyzing the TNR, ACC, TPR, and NAPR performance metrics.

6.5.2.2.4.2 Detailed Results

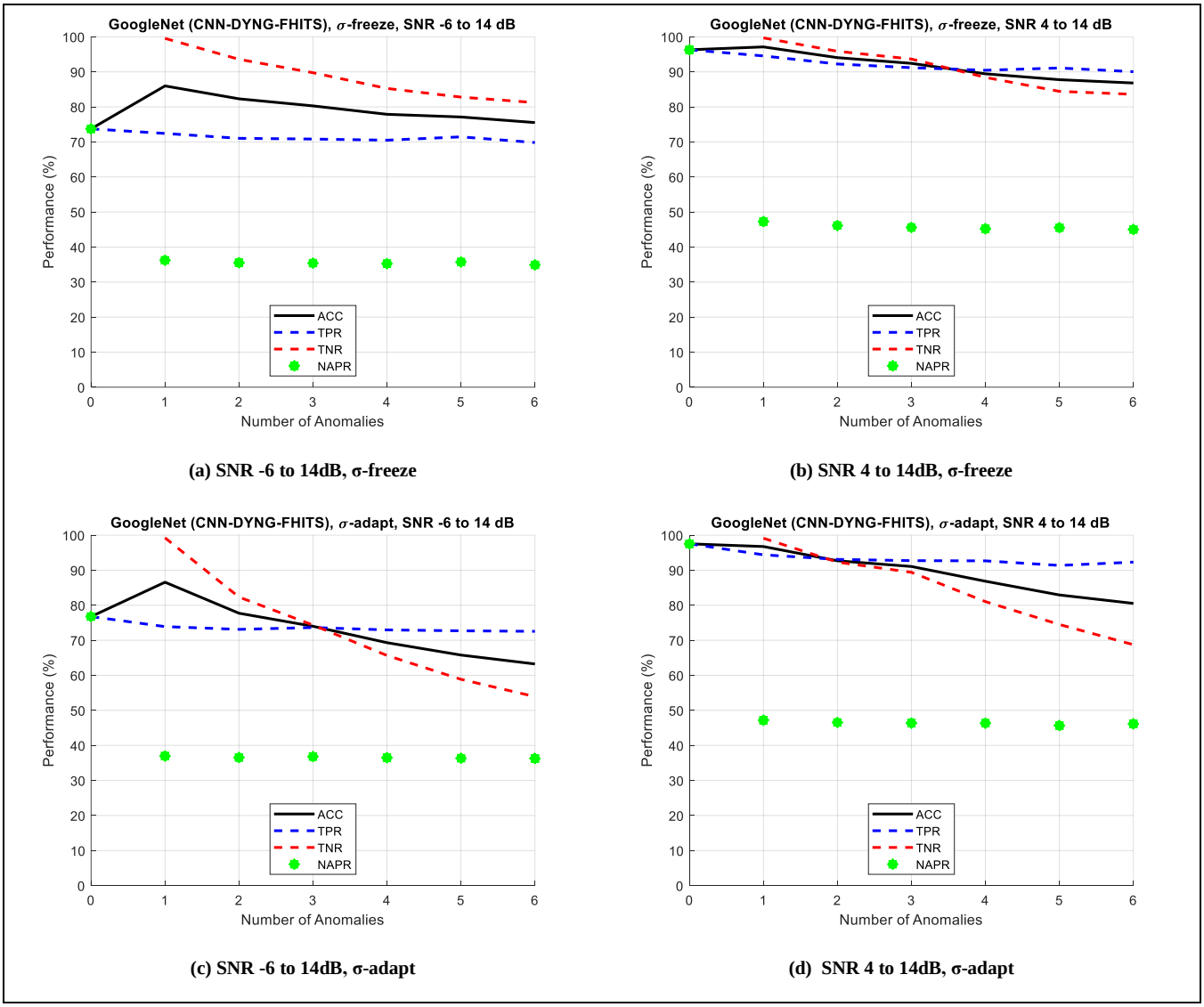
This section summarizes the more detailed results of the DYNG-FHITS algorithm applied to the different CNNs studied. In Figure 137 through Figure 141, sub-figures (a) and (b) show results with σ-frozen with lower and higher SNR levels respectively , while sub-figures (c) and (d) show results with *σ*-adapt with lower and higher SNR levels respectively.

In all cases, the overall accuracy (ACC) of the higher SNR levels show greater performance than the lower SNR levels. Promising results show that the accuracy of detecting the known waveforms (TPR) in all figures has some degradation as the number of anomalies increases, however in general good performance is maintained above non-adaptive positive rate (NAPR). This is an important and significant result showing that the original accuracy of the non-adaptive classifier is generally maintained by the DYNG-FHITS adaptation even in the presence of anomalies.

The accuracy of detecting anomalies (TNR) shows performance of the anomaly detection and clustering ability. In all cases, as the number of anomalies increases, there is a tendency for TNR to degrade, however in all cases TNR is maintained above the NAPR rate.

With respect to σ-frozen or adapting, in general one can see that keeping σ-frozen tends to provide the best and most stable performance overall. This is an unexpected outcome, because σ-

adapt is expected to capture the true variability of the anomalous data on the fly, and therefore expected to provide greater performance than σ-frozen. This outcome requires further research to explain the discrepancy.



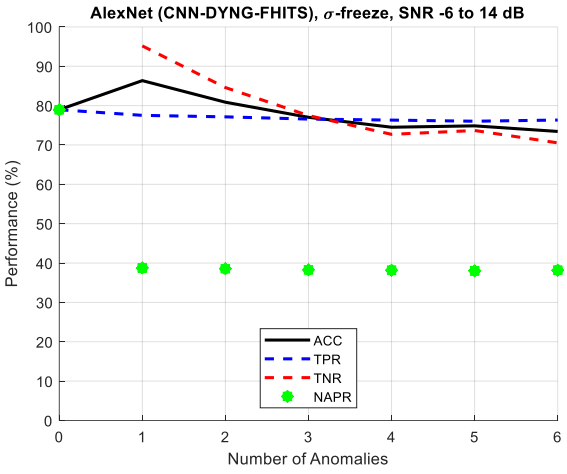
(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

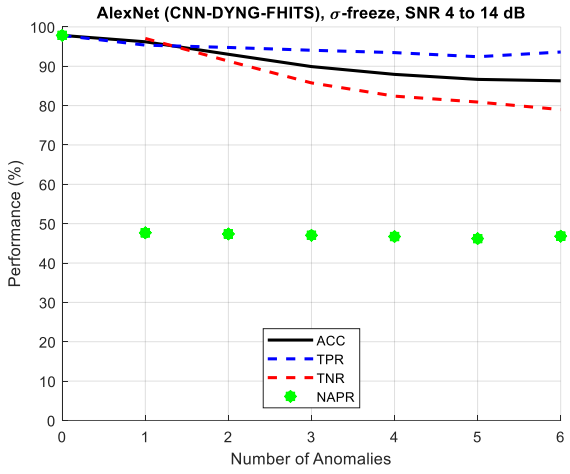
(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

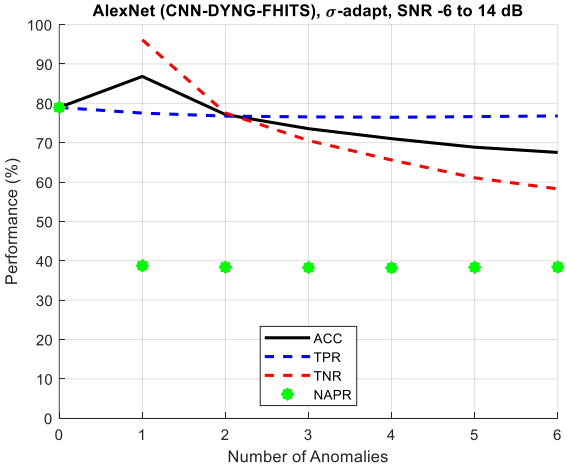
Figure 137. DYNG-FHITS GoogleNet Performance



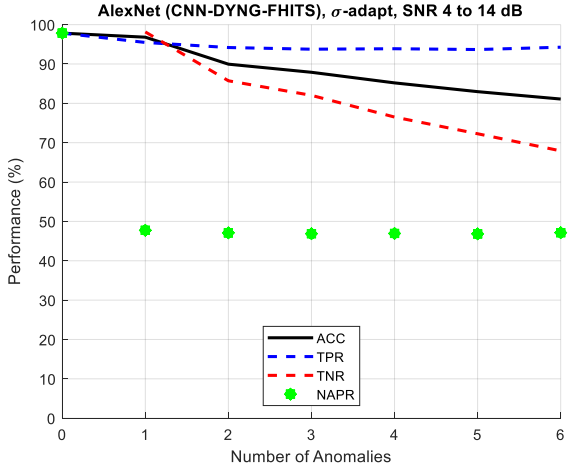
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

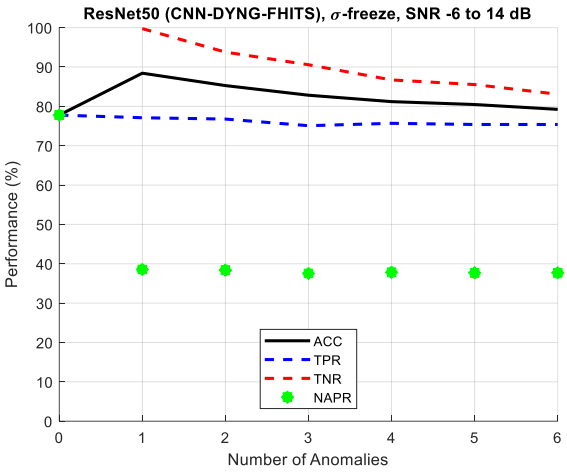


(c) SNR -6 to 14dB, σ-adapt

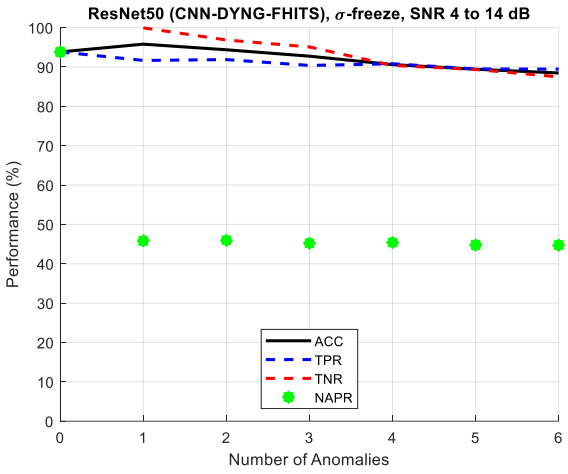


(d) SNR 4 to 14dB, σ-adapt

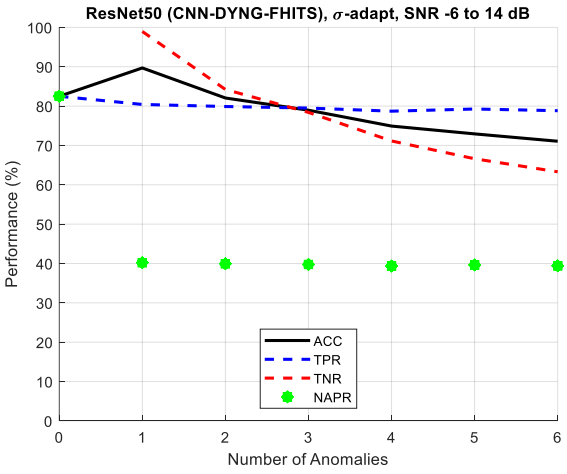
Figure 138. DYNG-FHITS AlexNet Performance



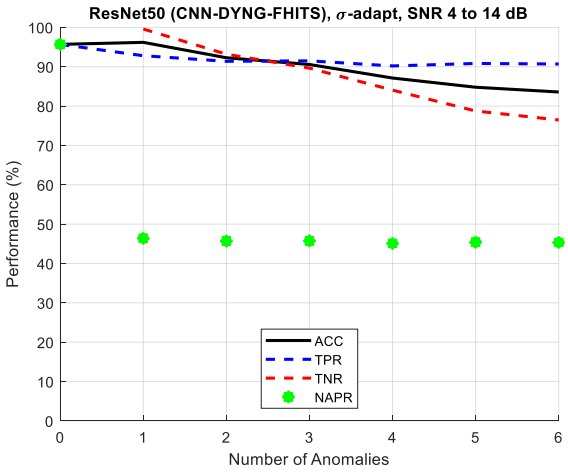
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

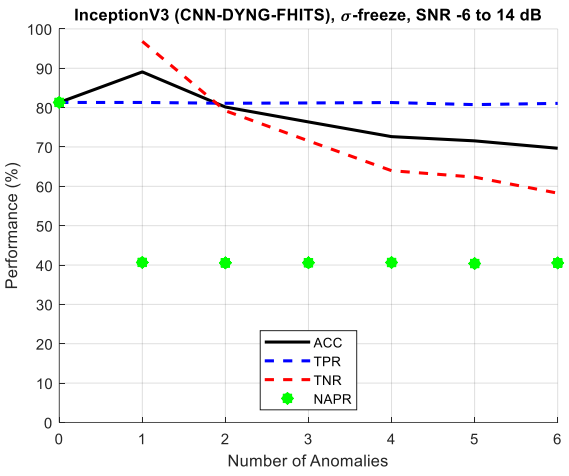


(c) SNR -6 to 14dB, σ-adapt

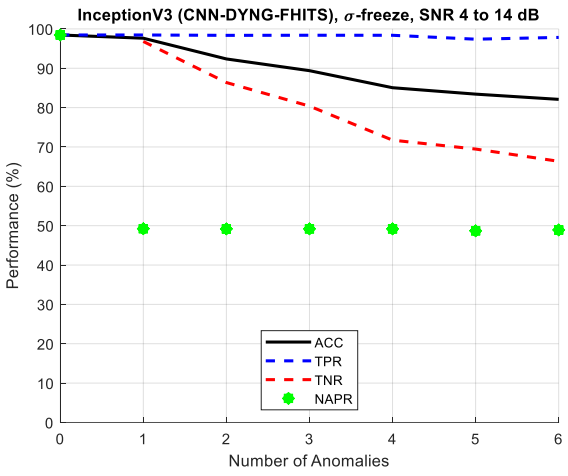


(d) SNR 4 to 14dB, σ-adapt

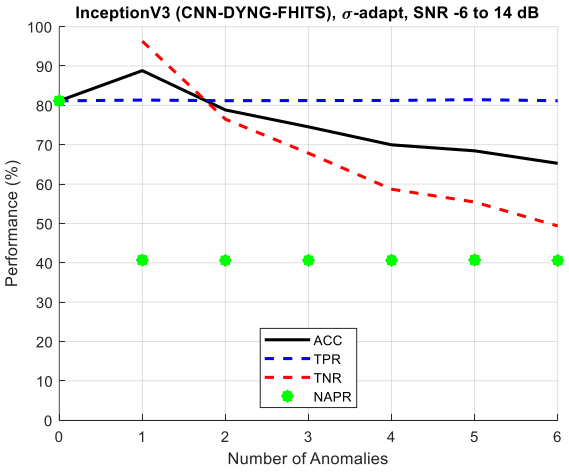
Figure 139. DYNG-FHITS ResNet50 Performance



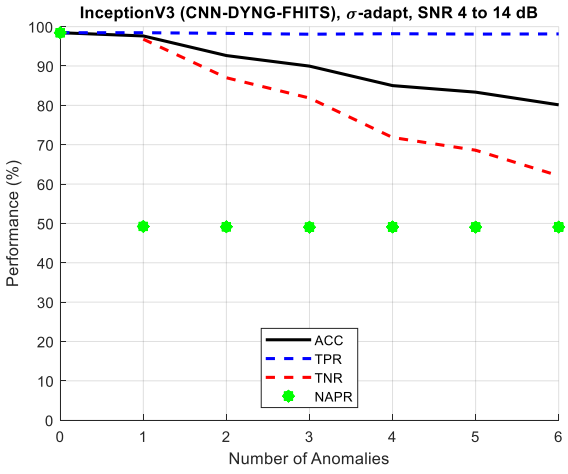
(a) SNR -6 to 14dB, σ-freeze



(b) SNR 4 to 14dB, σ-freeze

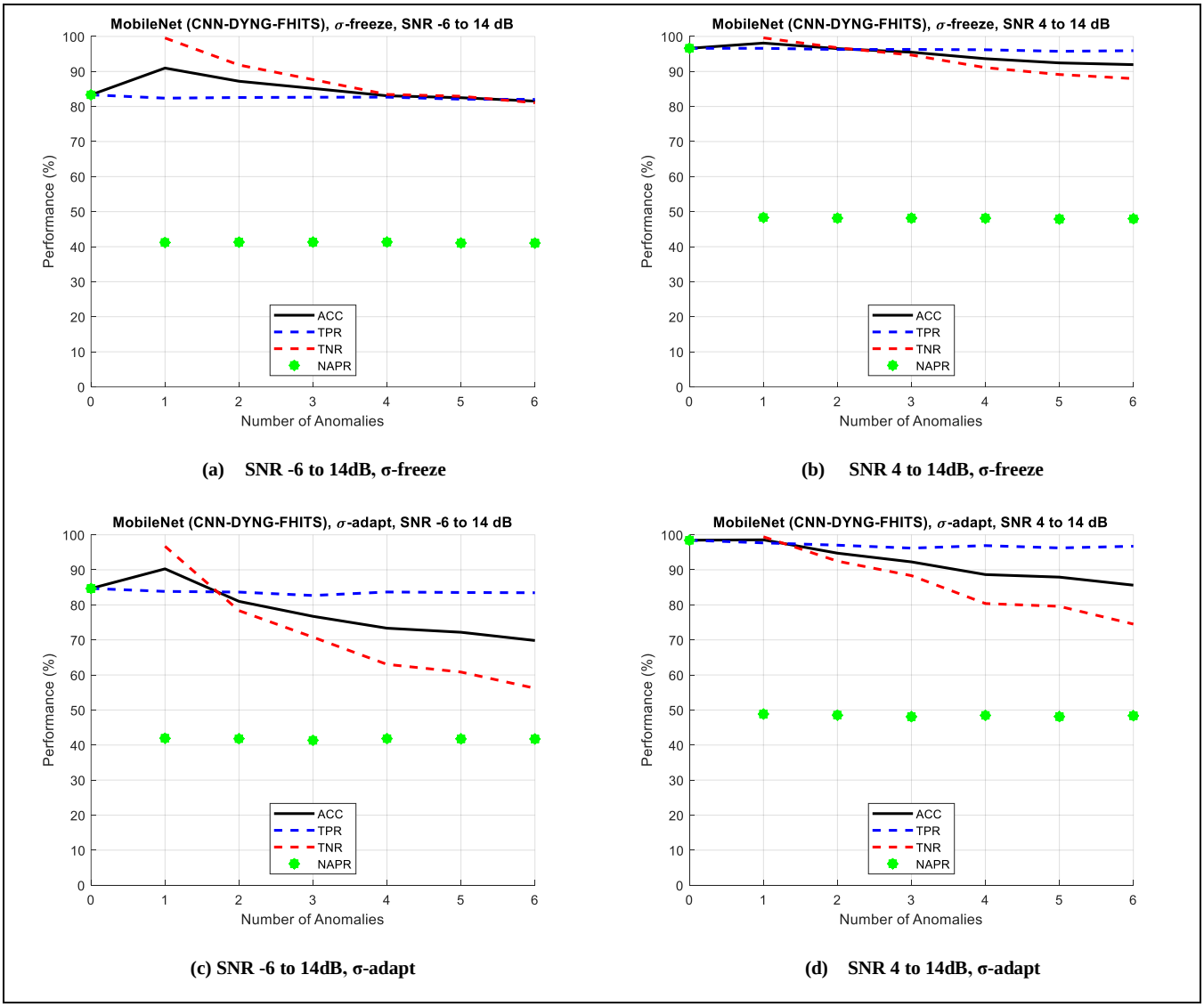


(c) SNR -6 to 14dB, σ-adapt



(c) SNR 4 to 14dB, σ-adapt

Figure 140. DYNG-FHITS InceptionV3 Performance



(a) SNR -6 to 14dB, σ-freeze

(b) SNR 4 to 14dB, σ-freeze

(c) SNR -6 to 14dB, σ-adapt

(d) SNR 4 to 14dB, σ-adapt

Figure 141. DYNG-FHITS MobileNet Performance

6.5.2.2.5 Comparative Summary

This section provides an overall performance comparison of the adaptive algorithms, with Table 41 and Table 42 summarizing results for the high and low SNR levels respectively, where the performance parameters across all 0 through 6 anomaly cases averaged into a single value to establish a high-level metric.

In each of the tables, the *Adaptive Algorithms* column lists the names of the algorithms under test, the *σ-freeze* and *σ-adapt* columns lists the performance results when the *σ* is frozen or adapts during online processing, the *ACC, TPR*, and *TNR* columns provide results from averaging the respective values calculated during the 0 through 6 anomaly test cases discussed in the previous sections, the *NAPR* column defines the non-adaptive prediction rate (accuracy that would have been obtained without adaptation), and finally the *ACC-DIFF* column shows the difference between the *ACC* columns for *σ-adapt* and *σ-freeze*, with a negative value indicating that σ-adapt causes degradation as compared to σ-freeze.

Table 41. Adaptive Performance Summary, High SNR 4 to 14 dB

Adaptive Algorithms	**σ-freeze % Performance**			**σ-adapt % Performance**			**Non adaptive positive rate**	**ACC of σ-adapt minus σ-freeze**
Name	**ACC**	**TPR**	**TNR**	**ACC**	**TPR**	**TNR**	**NAPR**	**ACC-DIFF**
FHITS-GoogleNet	93.2	96.9	88.4	90.8	97.8	81.4	55.6	-2.4
FHITS-AlexNet	87.4	97.5	73.9	85.5	97.5	69.5	55.7	-1.9
FHITS-ResNet50	94.9	97.2	91.8	92.6	97.5	86.0	55.6	-2.3
FHITS-InceptionV3	81.2	98.2	58.6	87.4	98.2	73.1	56.1	6.2
FHITS-MobileNet	93.8	97.6	88.9	91.9	98.2	83.5	55.9	-1.9
CADC-GoogleNet	84.9	95.7	70.3	80.3	96.0	59.3	54.8	-4.6
CADC-AlexNet	84.0	94.1	70.5	79.3	94.1	59.4	53.8	-4.7
CADC-ResNet50	89.4	95.1	81.8	85.1	95.1	71.9	54.3	-4.3
CADC-InceptionV3	67.3	96.9	27.8	64.2	96.9	20.6	55.4	-3.1
CADC-MobileNet	79.1	96.3	56.0	75.9	96.3	48.7	55.0	-3.2
DYNG-FHITS-GoogleNet	92.0	92.3	90.9	89.8	93.4	84.2	53.4	-2.2
DYNG-FHITS-AlexNet	91.1	94.5	86.1	88.8	94.7	80.4	54.3	-2.3
DYNG-FHITS-ResNet50	92.2	91.1	93.2	90.0	91.9	86.9	52.5	-2.2
DYNG-FHITS-InceptionV3	89.8	98.2	78.5	89.6	98.2	78.0	56.2	-0.2
DYNG-FHITS-MobileNet	95.0	96.2	93.2	92.3	97.0	85.8	55.3	-2.7
DYNG-CADC-GoogleNet	85.6	93.6	74.4	84.1	93.7	70.7	53.8	-1.5
DYNG-CADC-AlexNet	85.9	96.3	71.9	85.4	96.0	71.0	55.1	-0.5
DYNG-CADC-ResNet50	90.2	94.6	83.9	90.8	94.3	85.6	54.2	0.6

DYNG-CADC-InceptionV3	68.9	98.0	29.9	68.0	98.2	27.8	56.1	-0.9
DYNG-CADC-MobileNet	81.3	97.7	59.4	80.5	97.6	57.6	55.9	-0.8

As an example of the high SNR results in Table 41, for FHITS-GoogleNet the overall accuracies were 93.2% and 90.8% in the σ-freeze and σ-adapt modes respectively, with an *ACC-DIFF* value -2.4%, indicating that σ-adapt performs worse than σ-freeze. As indicated by the mostly negative values throughout the *ACC-DIFF* column, σ-adapt mode performs more poorly than σ-freeze, however an exception is FHITS-InceptionV3, where σ-adapt provides a significant improvement of 6.2% over σ-freeze.

Table 42. Adaptive Performance Summary, Low SNR -6 to 14 dB

Adaptive Algorithms	**σ-freeze Performance**			**σ-adapt Performance**			**Non adaptive positive rate**	**ACC of σ-adapt minus σ-freeze**
Name	**ACC**	**TPR**	**TNR**	**ACC**	**TPR**	**TNR**	**NAPR**	**ACC-DIFF**
FHITS-GoogleNet	83.2	80.8	86.5	76.7	81.5	70.2	46.4	-6.5
FHITS-AlexNet	72.8	81.8	60.6	72.5	81.8	60.1	46.8	-0.3
FHITS-ResNet50	84.8	82.9	87.4	80.6	84.2	75.8	47.8	-4.2
FHITS-InceptionV3	71.5	85.0	53.5	75.9	85.0	63.8	48.6	4.4
FHITS-MobileNet	83.6	85.2	81.5	78.7	85.9	69.0	48.9	-4.9
CADC-GoogleNet	71.2	75.2	65.7	65.5	76.4	51.0	43.4	-5.7
CADC-AlexNet	68.9	70.2	67.2	63.4	70.2	54.4	40.1	-5.5
CADC-ResNet50	77.4	81.4	72.0	71.8	81.5	58.8	46.6	-5.6
CADC-InceptionV3	57.8	81.8	25.6	54.3	81.8	17.5	46.8	-3.5
CADC-MobileNet	71.5	82.2	57.1	66.2	82.3	44.6	47.1	-5.3
DYNG-FHITS-GoogleNet	79.0	71.4	88.7	73.3	73.7	72.4	41.7	-5.7
DYNG-FHITS-AlexNet	78.0	77.0	79.0	74.8	77.1	71.5	44.2	-3.2
DYNG-FHITS-ResNet50	82.2	76.2	89.9	78.9	79.9	77.1	44.7	-3.3
DYNG-FHITS-InceptionV3	77.3	81.1	72.0	75.3	81.3	67.4	46.4	-2.0
DYNG-FHITS-MobileNet	84.8	82.5	87.7	78.3	83.6	71.0	47.6	-6.5
DYNG-CADC-GoogleNet	70.9	73.4	67.8	71.0	74.3	65.7	42.4	0.1
DYNG-CADC-AlexNet	74.0	77.8	68.8	73.6	77.7	68.0	44.5	-0.4
DYNG-CADC-ResNet50	77.4	81.0	72.2	78.1	80.9	73.8	46.4	0.7
DYNG-CADC-InceptionV3	57.1	80.0	26.2	55.9	79.9	23.8	45.8	-1.2
DYNG-CADC-MobileNet	72.6	82.7	59.2	72.3	83.1	57.7	47.4	-0.3

The results in Table 42 indicate that overall lower SNR signals cause degradation as compared to the higher SNR levels shown in Table 41 as indicated by the mostly negative numbers in the *ACC_DIFF* column, and consistent with the higher results in Table 41, σ-adapt mode generally causes degradation as compared to σ-freeze.

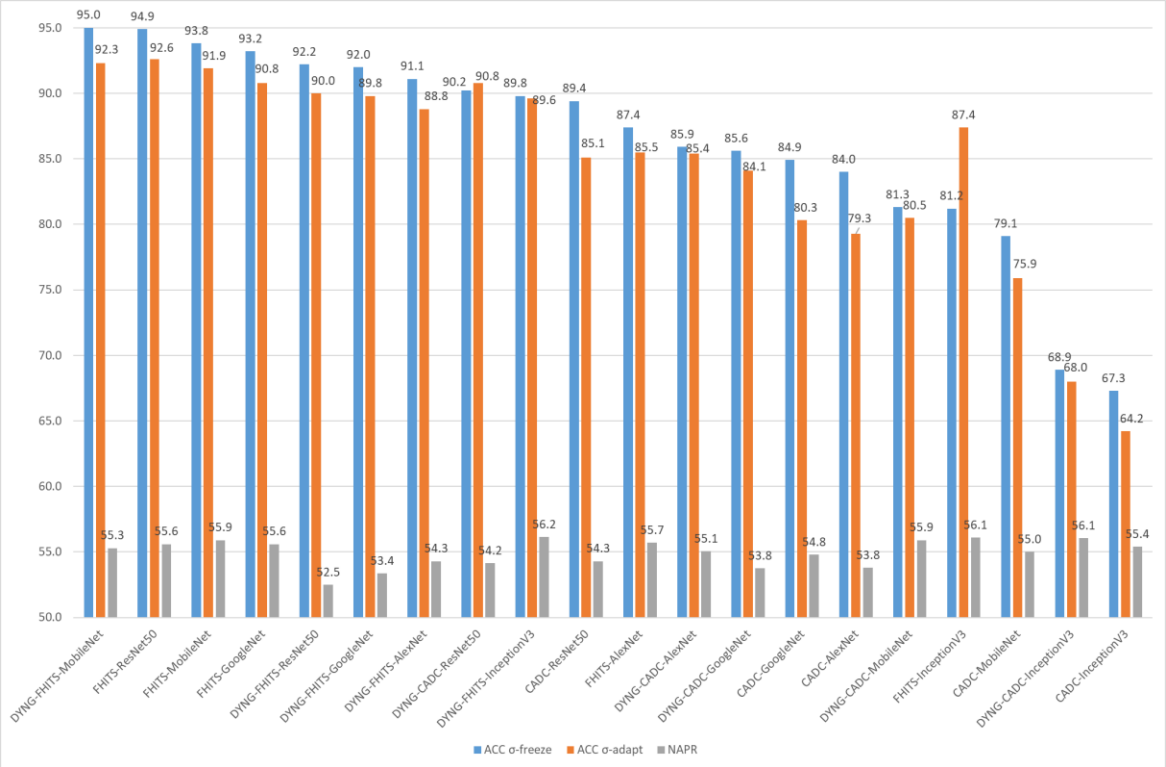


Figure 142. Overall Adaptive Accuracies, σ-freeze and σ-adapt, High SNR 4 to 14 dB

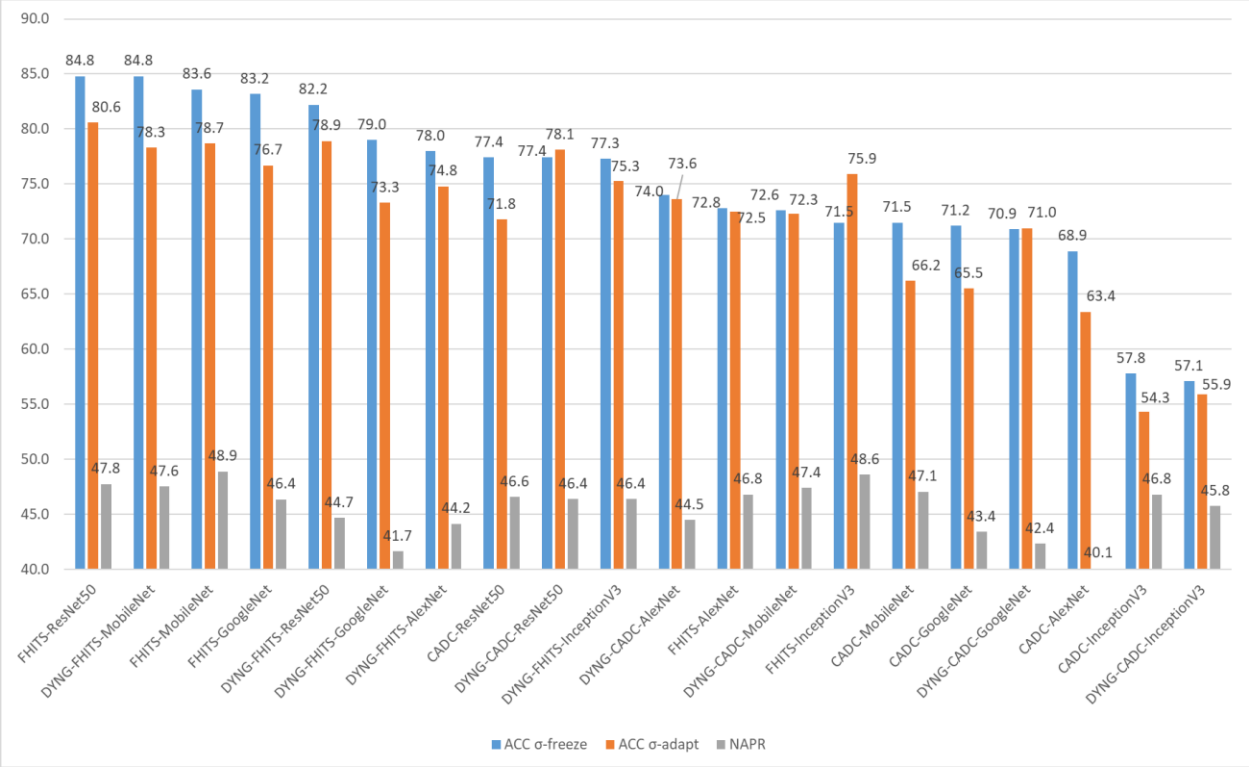


Figure 143. Overall Adaptive Accuracies, σ-freeze and σ-adapt, Low SNR -6 to 14 dB

Figure 142 and Figure 143 provides a visual summary of the overall accuracies for the σ-freeze and σ-adapt modes tested with high and low SNR data ranges respectively, and in both cases in general, the σ-freeze mode generally provides better performance than σ-adapt in most of the algorithms, except for FHITS-InceptionV3 and DYNG-CADC-ResNet50. Finally, it is important to note that in all cases, the overall performance is well above the NAPR.

In two especially important cases that were more obvious in the detailed results sections, the TNR and ACC values remained relatively stable across all anomalies introduced for the DYNG-FHITS with ResNet50 and GoogleNet for the low and high SNR levels, and when keeping σ-freeze. These results may suggest that the selected hyper parameters were closer to the optimal for those two algorithms and therefore provided a higher level of performance. As a reminder, all hyper parameters during the experiments were chosen to give some reasonable level of performance,

however they were not selected to provide optimal performance as doing so would have increased the complexity of the experiments.

To summarize, the results show that all the adaptive algorithms CADC, FHITS, DYNG-CADC, and DYNG-FHITS show promise in sustaining reasonable classification performance while operating online in non-stationary RF environments. In general, σ-freeze tends to provide the best performance despite the expectation that σ-adapt would provide for better performance, as σ-adapt is presumed to take into account the actual streaming anomalous data samples instead of relying on prior σ initialization values which were based on training from the known data classes. It is speculated that because the σ-adapt mode operates with one-shot sampling updates, that improvement may be obtained by collecting and storing a small set of n anomalous samples and using them to perform the update, as this may allow for acquiring greater statistical information before performing the updates. As such, the σ-adapt mode warrants further research.

7 SUMMARY

Of most importance to this research was the development of one-shot adaptive classification algorithms to perform online classification, anomaly detection, and adaptive classification on streaming waveform data in non-stationary environments. The major outcome of this effort was the development of the CADC, FHITS, DYNG-CADC, and DYNG-FHITS algorithms. The algorithms are of low computational complexity, therefore making them viable solutions for real-time streaming data applications. These adaptive algorithms when applied to classify several digital waveforms maintain acceptable classification accuracies in non-stationary RF environments. In two important cases (σ-freeze with DYNG-FHITS with ResNet50 or GoogleNet), the ACC performance remained very stable at 90% or greater across all anomalies introduced at the high SNR levels. For lower SNR levels, σ-freeze with DYNG-FHITS with ResNet50 gave overall accuracy of performance greater than 80%.

Additionally, this research explored extending the functionality of CNNs by replacing the last classification layer of the networks with the CADC, FHITS, DYNG-CADC, and DYNG-FHITS adaptive classifiers, allowing the CNN architectures to maintain performance accuracies in non-stationary environments. In general, poor classification performance results occurred primarily due to low SNR waveforms.

An unexpected result of the CADC and FHITS algorithms was that σ-freeze mode generally outperformed σ-adapt. Intuitively, the adaptation process in σ-adapt should have allowed for capturing of the actual variances of the anomalous data, and therefore more accurately modeled the data. An explanation for this counter-intuitive phenomenon is that σ-adapt adjusts σ too quickly

when not enough samples are in the cluster. The presence of two consecutive anomalous data points that are close to each other can cause σ to reduce very quickly. To resolve this, a cluster density value or counter could be maintained, and this value could be compared against a threshold to determine if σ should be adjusted. Another is possibly that the hyperparameter η that controls the spread of σ-adapt may not be optimally set, therefore causing misclassifications.

This research also explored using transfer learning with CNNs pretrained on ImageNet images as feature extractors for RF waveform classification. The results showed that (i) the constellation image generation technique that was applied captures enough details of each modulation at all SNRs tested and (ii) the features captured from the images are rich enough to differentiate distinguishing characteristics in the constellation images of RF waveforms.

Preliminary works also demonstrated that normal distribution modeling with feature compression using PCA can effectively perform anomaly detection for CNN classifiers. This work also demonstrated that the development of a novel Random M-Class Anomaly detector for clustering anomalous waveform data could achieve a 95% accuracy, however this approach is not practical for online classification and adaptation because the initial assignment of the anomaly detector weights must rely on information obtained from the anomaly stimulus, rather than from a random assignment process, to quickly establish a reliable detector.

8 FUTURE WORK

The listing below provides an outline of the many areas for future work relevant to this research.

- Explore method(s) for selecting the optimal hyper-parameters for the adaptive algorithms
- Explore the characteristics required of feature vectors to provide for optimal performance and adaptation stability *(as was displayed DYNG-FHITS with ResNet50 or GoogleNet)*.
- Investigate σ-freeze verses σ-adapt, improve σ-adapt performance. Instead of adaptive based on one-shot, consider adapting based on n samples.
- Use a larger and more diverse waveform dataset, for example introduce real atmospheric effects such phase shifts and fading, and or use real RF waveforms acquired from actual systems.
- Perform a tradeoff study comparing the adaptive algorithms with different types of waveform feature sets, for example features generated from CNN architectures compared to features generated from the raw waveform data using HOS, DNN, CNN, and other feature generation techniques.
- Investigate incorporating concept drift techniques within the CADC and FHITS algorithms to make them more robust in more complex environments.
- Regarding anomaly detection, consider using the normal distribution PCA anomaly detector as a "precursor" anomaly detector (PCA model trained on the known waveforms), and then when a stimuli is flagged as an anomaly by the precursor anomaly detector, pass the stimuli to the adaptive algorithms (CADC and FHITS) and use their internal mechanisms for adaptive clustering and anomaly detection. Because the PCA detector relies on using prior information from the training data,

this approach may help to reduce known stimuli from being classified into unknowns.

- Throughout this research, the mean, standard deviation, and L2 norm metrics were used for representing the class centroid, however other metrics can be explored such as using the median, and median absolute deviation (MAD), as these are particularly useful to enhance performance in noisy environments[108].
- Implement adaptive algorithms on an actual cognitive RF testbed to demonstrate cognitive abilities
    - Test Example - in an adaptive frequency hopping application a test scenario could be to use the adaptive classification algorithms in concert with a RF spectrum occupancy prediction algorithm, to track and predict the behavior of a targeted frequency hopping emitter.
- For the Random M-class anomaly detector, investigate a systematic way of picking the randomly generated weight and bias vectors such that they fall away from or between the existing known classes. A candidate algorithm could be to compute the distance (Euclidean or other) between all combinations of two known classes and note the minimum and maximum values. Then while randomly creating new weight and bias vectors, constrain them to fall no closer than the minimum distance of all known classes, or to fall greater than the maximum distance. This technique may make the Random M-class anomaly detector more practical because the now the set of random weights and biases do not overlap with the space of known classes, and therefore prevents degradation of the original classifier performance on known classes.

9 REFERENCES

[1] O. Beyer, "Life-long Learning with Growing Conceptual Maps," Doctor of Philosophy Doctoral, Computer Science, Technische Fakulat der Universitat Bielefeld, Bielefeld, Germany, 2013.

[2] F. S. Kamps, C. L. Hendrix, P. A. Brennan, and D. D. Dilks, "Connectivity at the origins of domain specificity in the cortical face and place networks," *Proc Natl Acad Sci U S A,* vol. 117, pp. 6163-6169, Mar 17 2020.

[3] F. Liu, C. Masouros, A. P. Petropulu, H. Griffiths, and L. Hanzo, "Joint Radar and Communication Design: Applications, State-of-the-Art, and the Road Ahead," *IEEE Transactions on Communications,* vol. 68, pp. 3834-3862, 2020.

[4] A. F. Martone, K. I. Ranney, K. Sherbondy, K. A. Gallagher, and S. D. Blunt, "Spectrum Allocation for Noncooperative Radar Coexistence," *IEEE Transactions on Aerospace and Electronic Systems,* vol. 54, pp. 90-105, 2018.

[5] Y.-J. Tang, Q.-Y. Zhang, and W. Lin, "Artificial Neural Network Based Spectrum Sensing Method for Cognitive Radio," *6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM).* 2010.

[6] D. B. Schuster. (2015) International Telecommunication Union - 150 Years of History: Adaptation to Change and the Opportunity for Reform. *IEEE Communications Magazine — Communications Standards Supplement* 6.

[7] S. Bhattarai, J.-M. Park, B. Gao, K. Bian, and W. Lehr, "An Overview of Dynamic Spectrum Sharing: Ongoing Initiatives, Challenges, and a Roadmap for Future Research," *IEEE Transactions on Cognitive Communications and Networking,* vol. 2, pp. 110-128, 2016.

[8] G. D. Menghwar and C. F. Mecklenbrauker, "Cooperative versus Non-cooperative Communications," presented at the 2nd International Conference on Computer, Control and Communication, Karachi, Pakistan, 2009.

[9] U. S. G. A. Office, "Internet of Things : FCC Should Track Growth to Ensure Sufficient Spectrum Remains Available," 2017.

[10] A. Kott, A. Swami, and B. J. West, "The Internet of Battle Things " *IEEE Computer Society,* vol. 49, p. 6, 2016.

[11] J. Lundén, "Spectrum sensing for cognitive radio and radar systems," (Doctoral dissertation), Helsinki University of Technology, 2009.

[12] B. Paul, A. R. Chiriyath, and D. W. Bliss, "Survey of RF Communications and Sensing Convergence Research," *IEEE Open Access,* vol. 5, pp. 252-270, 2017.

[13] P. Steenkiste, D. Sicker, G. Minden, and D. Raychaudhuri, "Future Directions in Cognitive Radio Network Research Future Directions in Cognitive Radio Network Research NSF Workshop Report," 2009.

[14] F. C. Commisison, "Spectrum Policy Task Force," *ET Docket No. 02- 135,* November 2002 2002.

[15] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE Journal on Selected Areas in Communications,* vol. 23, pp. 201-220, 2005.

[16] A. Chaudhari and D. Squires, "Colosseum: A Battleground for AI Let Loose on the RF Spectrum," *Microwave Journal,* p. 10, 2018.

[17] A. F. Martone. (2014, September 2014) Cognitive Radar Demystified. *Radio Science Bulletin*. 13.

[18] A. Abdelmutalab, K. Assaleh, and M. El-Tarhuni, "Automatic modulation classification based on high order cumulants and hierarchical polynomial classifiers," *Physical Communication,* vol. 21, pp. 10-18, 2016.
[19] D.-C. Chang and P.-K. Shih, "Cumulants-based modulation classification technique in multipath fading channels," *IET Communications,* vol. 9, pp. 828-835, 2015.
[20] C. C. L. C. Freitas, F. C. B. F. Muller, J. W. A. Costa and A. Klautau, "Automatic Modulation Classification for Cognitive Radio Systems: Results for the Symbol and Waveform Domains," presented at the IEEE Latin-American Conference on Communications, Medellin, Colombia, 2009.
[21] M. W. Aslam, Z. Zhu, and A. K. Nandi, "Autmomatic Modulation Classification Using Combination of Genetic Programming and KNN," *IEEE Transactions on Wireless Communications,* pp. 1-9, 2012.
[22] S. Peng, H. Jiang, H. Wang, H. Alwageed, and Y.-D. Yao, "Modulation classification using convolutional Neural Network based deep learning model," in *26th Wireless and Optical Communication Conference*, 2017.
[23] X. Zhu, Goldberg A.B., "Introduction to Semi-supervised Learning," *Morgan & Claypool Publishers,* p. 130, 2009.
[24] T. Lu, G. Liu, W. Li, S. Chang, and W. Guo, "Distributed sampling rate allocation for data quality maximization in rechargeable sensor networks," *Journal of Network and Computer Applications,* vol. 80, pp. 1-9, 2017.
[25] X. Wei, Y. Liu, X. Wang, S. Gao, and L. Chen, "Online Adaptive Approximate Stream Processing With Customized Error Control," *IEEE Access,* vol. 7, pp. 25123-25137, 2019.
[26] A. Bifet and G. D. F. Morales, "Big Data Stream Learning with SAMOA," *International Conference on Data Mining Workshop,* pp. 1199-1202, 2014.
[27] B. Krawczyk, B. Pfahringer, and M. Wo´zniak, "Combining Active Learning with Concept Drift Detection for Data Stream Mining," *2018 IEEE International Conference on Big Data (Big Data),* p. 6, 2018.
[28] S.-H. Cha, "Comprehensive Survey on Distance/Similarity Measures between Probability Density," *International Journal of Mathematical Models and Methods in Applied Sciences,* April 9, 2007.
[29] U. Bud and J. Lim, "Distance Functions to Detect Changes in Data Streams," *International Journal of Information Processing Systems,* vol. 2, p. 4, 2006.
[30] D. H. Tran, "Change Detection in Streaming Data," *Dissertation,* 2013.
[31] R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification," *John Wiley & Sons,* 2001.
[32] S. Russell and P. Norvig, "Artificial Intelligence, A Modern Approach," *Pearson Education, Inc.*, 2010.
[33] N. M. Varma and A. Choudhary, "Evaluation Of Distance Measures In Content Based Image Retrieval," in *Third International Conference on Electronics Communication and Aerospace Technology*, 2019, p. 6.
[34] M. Hazewinkel, "Bhattacharyya distance," *Encyclopedia of Mathematics, Springer Science Business Media B.V. / Kluwer Academic Publishers,* 2001.
[35] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal,* vol. XXVII, p. 45, 1948.
[36] R. C. Gonzalez and R. E. Woods, "Digital Image Processing 3rd ed.," *Pearson Printice Hall, Inc.*, 2008.
[37] D. Hong, D. Zhao, and Y. Zhang, "The Entropy and PCA Based Anomaly Prediction in Data Streams," *Procedia Computer Science,* vol. 96, pp. 139-146, 2016.
[38] S. L. Kullback, R. A., "On Information and Sufficiency," *The Annals of Mathematical Statistics,* vol. 22, p. 9, 1951 1951.

[39] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Fourth ed.: Cambridge University Press, 2003.

[40] F. Nielsen and R. Nock, "Entropies and Cross-Entropies of Exponential Families," in *17th International Conference on Image Processing*, Hong Kong, 2010, p. 4.

[41] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*: Pearson Addison Wesley, 2006.

[42] E.E. Azzouz and A. K. Nandi, "Automatic identification of digital modulation types," *Elsevier Signal Processing*, vol. 47, p. 19, 1994.

[43] C. Cortes and V. Vapnik, "Support-Vector Networks," presented at the Machine Learning, Boston, 1995.

[44] X. Zhou, Y. Wu, and B. Yang, "Signal Classification Method Based on Support Vector Machine and High-Order Cumulants," *Wireless Sensor Network*, vol. 02, pp. 48-52, 2010.

[45] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*. Cambridge Massachusetts: MIT Press, 2000.

[46] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, *et al.*, *Neural Networks and Learning Machines*, 2014.

[47] M. Shah and P. R. Kapdi, "Object Detection Using Deep Neural Networks," in *International Conference on Intelligent Computing and Control Systems*, Madurai, India, 2017, p. 4.

[48] A. Elhassouny and F. Smarandache, "Trends in Deep Convolutional Neural Network Architectures: A Review," presented at the International Conference of Computer Science and Renewable Energies (ICCSRE), Agadir, Morocco, 2019.

[49] N. E. West and T. O'Shea, "Deep Architectures for Modulation Recognition," in *IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN*, Piscataway, NJ, USA, 2017.

[50] D. Soekhoe, P. v. d. Putten, and A. Plaat. (2016, On the Impact of Data Set Size in Transfer Learning using Deep Neural Networks. *Boström H., Knobbe A., Soares C., Papapetrou P. (eds) Advances in Intelligent Data Analysis XV. IDA 2016. Lecture Notes in Computer Science 9897,* 12.

[51] A. Canziani , E. Culurciello, and A. Paszke. (2017, An Analysis of Deep Neural Network Models for Practical Applications. *Computer Vision and Pattern Recognition*. Available: arXiv:1605.07678v4

[52] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," presented at the IEEE Conference on Computer Vision and Pattern Recognition, LKas Vegas, NV, 2016.

[53] C. Szegedy, Y. J. Wei Liu, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, *et al.*, "Going Deeper with Convolutions," *Conference on Computer Vision and Pattern Recognition*, 2015 2015.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," presented at the Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[55] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 14, 2019.

[56] S. A. Israel, J. H. Goldstein, J. S. Klein, J. Talamonti, F. Tanner, S. Zabel, *et al.*, "Generative Adversarial Networks for Classification," presented at the Applied Imagery Pattern Recognition Workshop (AIPR), 2017.

[57] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, *et al.*, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[58] Y. Hong, U. Hwang, J. Yoo, and S. Yoon. (2018, How Generative Adversarial Networks and Their Variants Work: An Overview. 41.

[59] J. Liu, M. Gong, and H. He, "Deep associative neural network for associative memory based on unsupervised representation learning," *Neural Netw*, vol. 113, pp. 41-53, May 2019.

[60] D. W. Yu Cheng, M. Pan Zhou, and T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding*, p. 10, 2020.

[61] A. M. Elbir and K. V. Mishra, "Joint Antenna Selection and Hybrid Beamformer Design using Unquantized and Quantized Deep Learning Networks," *IEEE Transactions on Wireless Communications*, vol. 19, p. 10, 23 November 2019 2019.

[62] S. Kauschke, D. H. Lehmann, and J. Furnkranz, "Patching Deep Neural Networks for Nonstationary Environments," presented at the International Joint Conference on Neural Networks, Budapest, Hungary, 2019.

[63] L. Fei-Fei, R. Fergus, and P. Perona, "One-Shot Learning of Object Categories," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006, p. 18.

[64] A. .G and H. M. Pandey, "Data Clustering Approaches Survey and Analysis," in *1st International Conference on Futuristic Trend in Computational Analysis and Knowledge Management (ABLAZE)*, 2015, p. 6.

[65] Y. Li, E. Schofield, and M. Gonen, "A Tutorial on Dirichlet Process Mixture Modeling," *J Math Psychol*, vol. 91, pp. 128-144, Aug 2019.

[66] M. Lotfi Shahreza, D. Moazzami, B. Moshiri, and M. R. Delavar, "Anomaly Detection Using a Self-Organizing Map and Particle Swarm Optimization," *Scientia Iranica*, vol. 18, pp. 1460-1468, 2011.

[67] D. Miljković, "Brief Review of Self-Organizing Maps," in *MIPRO 2017/CTS*, 2017.

[68] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, p. 17, 1990.

[69] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps," *Springer-Verlag*, p. 11, 25 July 1982 1982.

[70] B. Fritzke, "A Growing Neural Gas Network Learns Topologies," in *7th International Conference on Neural Information Processing Systems*, 1994, p. 8.

[71] T. Martinetzt and K. Schultent, "A Neural Network with Hebbian-like Adaptation Rules Learning Visuomotor Coordination of a PUMA Robot " *IEEE International Conference on Neural Networks*, 06 August 2002 1993.

[72] O. Beyer, "Online Semi-Supervised Growing Neural Gas," *International Journal of Neural Systems, World Scientific Publishing Company*, vol. 0, p. 15, 2000.

[73] O. Beyer, "DYNG: Dynamic Online Growing Neural Gas for Stream Data Classificatio," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013, p. 6.

[74] M. Goldstein and S. Uchida, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data," *PLoS One*, vol. 11, p. e0152173, 2016.

[75] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection for Discrete Sequences: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, pp. 823-839, 2012.

[76] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-Wide Traffic Anomalies," presented at the SIGCOMM 04, Portland Oregon, 2004.

[77] S. Jin and D. S. Yeung, "A Covariance Analysis Model for DDoS Attack Detection," *IEEE Communications Society*, p. 5, 2004.

[78] S. P. Trinita, Y. Purwanto, and T. W. Purboyo, "A Sliding Window Technique for Covariance Matrix to Detect Anomalies on Stream Traffic," in *International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, 2015, p. 6.

[79] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under Concept Drift: A Review," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1-1, 2018.
[80] Q. Sun, H. Liu, and T. Harada, "Online growing neural gas for anomaly detection in changing surveillance scenes," *Pattern Recognition*, vol. 64, pp. 187-201, 2017.
[81] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly Detection using One-Class Neural Networks," presented at the Knowledge Discovery and Data Mining Conference, London, United Kingdom, 2018.
[82] M. L. Skolnik, *Introduction to Radar Systems*, 2 ed.: McGraw-Hill Publishing Company, 1980.
[83] D. Z. B. P. Lathi, *Modern Digital and Analog Communication Systems*, Fourth ed. New York: Oxford University Press, 2010.
[84] R. Krishnan, R. G. Babu, S. Kaviya, N. P. Kumar, C. Rahul, and S. S. Raman, "Software Defined Radio (SDR) Foundations, Technology Tradeoffs: A Survey," *IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI-2017)*, 2017.
[85] J. R. Guerci, "Cognitive Radar: A Knowledge-Aided Fully Adaptive Approach," *2010 IEEE Radar Conference*, p. 6, 2010.
[86] J. Mitola, "Cognitive Radio: Making Software Radios More Personal," *IEEE Personal Communications*, vol. 6, p. 6, 4 Aug 1999 1999.
[87] D. M. Zasada, J. J. Santapietro, and L. D. Tromp, "Implementation of a Cognitive Radar Perception/Action Cycle," presented at the 2014 Radar Conference, 2014.
[88] G. E. Schrader, "The Knowledge Aided Sensor Signal Processing and Expert Reasoning (KASSPER) Real-Time Signal Processing Architecture," in *P2004 IEEE Radar Conference* Philadelphia, PA, USA,, 2004.
[89] S. Haykin. (2006, January 2006) Cognitive Radar : A Way of the Future. *IEEE Signal Processing Magazine*. 11.
[90] K. L. Bell, C. J. Baker, G. E. Smith, J. T. Johnson, and M. Rangaswamy, "Cognitive Radar Framework for Target Detection and Tracking," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, pp. 1427-1439, 2015.
[91] S. Bruggenwirth, M. Warnke, S. Wagner, and K. Barth, "Cognitive Radar for Classification," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, pp. 30-38, 2019.
[92] A. E. Wolke. (2014). *Basics of IQ Signals and IQ modulation & demodulation - A tutorial*. Available: https://youtu.be/h_7d-m1ehoY
[93] K. I.-T. Chiu, "Recovery of Symbol Sampling Time for North American Digital Cellular (NADC) System," Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1995.
[94] J. Renard, J. Verlant-Chenet, J. M. Dricot, P. De Doncker, and F. Horlin, "Higher-order cyclostationarity detection for spectrum sensing," *Eurasip Journal on Wireless Communications and Networking*, 2010.
[95] H. Saggar and D. K. Mehra, "Cyclostationary Spectrum Sensing in Cognitive Radios Using FRESH Filters," *Advances in Wireless Cellular Telecommunications: Technologies & Services, 1st ICEIT National Conference,arXiv:1312.5257 [cs.OH]*, 2011.
[96] T. Yucek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Surveys & Tutorials*, vol. 11, pp. 116-130, 2009.
[97] S. Ding, H. Zhu, W. Jia, and C. Su, "A survey on feature extraction for pattern recognition," *Artificial Intelligence Review*, vol. 37, pp. 169-180, 2011.
[98] S. M. Jameel, M. A. Hashmani, M. Rehman, and A. Budiman, "Adaptive CNN Ensemble for Complex Multispectral Image Analysis," *Complexity*, vol. 2020, pp. 1-21, 2020.

[99] A. S. Krizhevsky, Ilya;Hinton, Geoffrey E. , "ImageNet classification with deep convolutional neural networks," in *25th International Conference on Neural Information Processing Systems* 2012, p. 9.

[100] S. Peng, H. Jiang, H. Wang, H. Alwageed, Y. Zhou, M. M. Sebdani, *et al.*, "Modulation Classification Based on Signal Constellation Diagrams and Deep Learning," *IEEE Trans Neural Netw Learn Syst*, vol. 30, pp. 718-727, Mar 2019.

[101] M. Conn, K. M'Bale, and D. Josyula, "Multi-level metacognition for adaptive behavior," *Biologically Inspired Cognitive Architectures*, vol. 26, pp. 174-183, 2018.

[102] S. H. Hasanpour, M. Rouhani, M. Fayyaz, and M. Sabokrou. Lets keep it simple, Using Simple Architectures to Outperform Deeper and More Complex Architectures [Online]. Available: https://arxiv.org/abs/1608.06037

[103] T. J. C. O'Shea, Johnathan; Clancy, Charles T. Convolutional Radio Modulation Recognition Networks [Online]. Available: https://arxiv.org/abs/1602.04105

[104] M. A. Conn and J. Darsana, "Radio Frequency Classification and Anomaly Detection using Convolutional Neural Networks," presented at the 2019 IEEE Radar Conference (RadarConf), Boston, MA, USA,, 2019.

[105] T. Kudo, T. Morita, T. Matsuda, and T. Takine, "PCA-Based Robust Anomaly Detection Using Periodic Traffic Behavior," presented at the International Conference on Communications - 1st IEEE Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios (TRICANS), 2013.

[106] O. Beyer, "Life-long Learning with Growing Conceptual Maps," *Phd Thesis*, vol. Technische Fakultat der Universita Bielefeld, p. 156, 2013.

[107] Y. Quintana-Pacheco, D. Ruiz-Fernandez, and A. Magrans-Rico, "Growing Neural Gas approach for obtaining homogeneous maps by restricting the insertion of new nodes," *Neural Netw*, vol. 54, pp. 95-102, Jun 2014.

[108] G. George, R. M. Oommen, S. Shelly, S. S. Philipose, and A. M. Varghese, "A Survey on Various Median Filtering Techniques For Removal of Impulse Noise From Digital Image," in *Conference on Emerging Devices and Smart Systems* Tamilnadu, India, 2018, p. 4.

APPENDIX A: ADAPTIVE CLASSIFIER TEST PROCEDURE

The data sets used for testing the adaptive classifiers are described in greater detail in section 3.4 for the known waveforms and section 4.4.2 for the anomalous waveforms. To carry out this series of tests we used only the even dB levels -6, -4, … 12, 14.

To test the adaptive classifier algorithms, the test setup was designed to mimic anticipated online scenarios, where the algorithms receive inputs from random arrivals of known and unknown (new) waveforms to assess how they classify and adapt under such conditions. To carry out the experiment, multiple independent test runs were performed, with each run having a unique combination of test samples randomly drawn from the test data sets and then input to the classifiers. To begin, the first set of test runs were configured with samples of only the known waveforms, results from each run were collected, and then the results from each test run were averaged together. Then, test sets were generated consisting of a mixture of all the known types, and a set of samples with only one anomaly type were input to the classifiers, and the results from each test set were averaged together. This process was carried out until we had a set containing a mixture of all the knowns and all the six anomalies.

Table 43 summarizes the configurations of each of the test runs. The *Number of Anomalies* column shows the number of anomalies for each run, and in each run, there were always a sample subset consisting of some number of all the nine (9) known waveforms. The *Anomalies per Run* column shows the configuration of the number of anomalies for each run. For example, with *Number of Anomalies* = 0, the number of *Anomalies per Run* had *10* runs with no anomalies in the test sets as indicated by the ten (10) *_NONE_* configuration flags. As another example, with *Number of Anomalies* = 1, the number of *Anomalies per Run* had *6* runs with one (1) anomaly type

in the test sets as indicated by the six (6) configuration flags { GFSK, AM_SSB, WBFM, PAM4, CPFSK, AM_DSB}, where the first run contained only GFSK anomalies, the second run contained only AM_SSB anomalies, etc. As a final example, with *Number of Anomalies* = 2, the number of *Anomalies per Run* had *13* runs with two (2) anomaly types in the test sets as indicated by the 13 (13) configuration sets {GFSK, AM_SSB}, {GFSK, WBFM}, {GFSK,PAM4}, {GFSK, CPFSK}, {GFSK, AM_DSB}, {AM_SSB, WBFM}, {AM_SSB, PAM4}, {AM_SSB, CPFSK}, {AM_SSB, AM_DSB}, {WBFM, PAM4}, {WBFM, CPFSK}, {WBFM, AM_DSB}, and {PAM4, AM_DSB}, where the first run contained only GFSK and AM_SSB anomalies, the second run contained only GFSK and WBFM anomalies, etc. All these anomaly test sets were randomly mixed with the known types as they were input into the adaptive classifiers.

As part of testing, a range of high and low SNR levels were considered to assess performance under such conditions. This is reflected by the *High SNR Levels* and *Low SNR Levels* column headings. Within these columns are the test set configuration parameters n, p, a, k, r, and d, defined as:

- n = total number of negative (anomaly) samples to assess overall performance
- p = total number of positive (known) samples to assess overall performance
- a = the number of anomaly types *(not the number anomaly samples)*
- k = the number of known types *(not the number known samples)*
- r = number of runs to assess performance at the specified number of anomalies
- d =number of dB levels

For each of the runs, the above parameters are used to compute the number of samples per anomaly, the number of anomaly samples per dB level, the number of samples per known type, and the number of known samples per dB level. All these details are captured within Table 43 .

Table 43. Adaptive Tests - Setup Configurations

Number of Anomalies	Anomalies per Run	High SNR Levels (4 to 14 dB)	Low SNR Levels -6 to 14 dB	Comments
0	**Run# Anomaly Configuration** 1. _NONE_ 2. _NONE_ 3. _NONE_ 4. _NONE_ 5. _NONE_ 6. _NONE_ 7. _NONE_ 8. _NONE_ 9. _NONE_ 10. _NONE_	n=0 p=3240 a=0 k=9 r=10 d=6 Per run samples per anomaly n/r/a=0 anomaly samples per dB n/r/a/d=0 samples per known p/r/k=36 known samples per dB p/r/k/d=6	n=0 p=5940 a=0 k=9 r=10 d=11 Per run samples per anomaly n/r/a=0 anomaly samples per dB n/r/a/d=0 samples per known p/r/k=66 known samples per dB p/r/k/d=6	
1	**Run# Anomaly Configuration** 1. GFSK 2. AM_SSB 3. WBFM 4. PAM4 5. CPFSK 6. AM_DSB	n=1944 p=1944 a=1 k=9 r=6 d=6 Per run samples per anomaly n/r/a=324 anomaly samples per dB n/r/a/d=54* samples per known p/r/k=36 known samples per dB p/r/k/d=6	n=3564 p=3564 a=1 k=9 r=6 d=6 Per run samples per anomaly n/r/a=594 anomaly samples per dB n/r/a/d=54* samples per known p/r/k=66 known samples per dB p/r/k/d=11	-*note – for anomalous testing, although there were only 50 samples per dB available (per Table 9), 54 samples per dB were used. To achieve this, we double sampled 4 samples from the anomaly data, to maintain a 50% ratio of known to anomalous data.
2	**Run# Anomaly Configuration** 1. {GFSK, AM_SSB} 2. {GFSK, WBFM} 3. {GFSK, PAM} 4. {GFSK, CPFSK}	n=4212 p=4212 a=2 r=13 d=6 k=9	n=7722 p=7722 a=2 k=9 r=13 d=11	-

	5. {GFSK, AM_DSB} 6. {AM_SSB, WBFM} 7. {AM_SSB, PAM4} 8. {AM_SSB, CPFSK} 9. {AM_SSB, AM_DSB} 10. {WBFM, PAM4} 11. {WBFM, CPFSK} 12. {WBFM, AM_DSB} 13. {PAM4, AM_DSB}	Per run samples per anomaly n/r/a=162 anomaly samples per dB n/r/a/d=27 samples per known p/r/k=36 known samples per dB p/r/k/d=6	Per run samples per anomaly n/r/a=297 anomaly samples per dB n/r/a/d=27 samples per known p/r/k=66 known samples per dB p/r/k/d=6	
3	**Run# Anomaly Configuration** 1. {GFSK, AM_SSB, PAM4} 2. {GFSK, AM_SSB, CPFSK} 3. {GFSK, AM_SSB, AM_DSB} 4. {AM_SSB, WBFM, PAM4} 5. {AM_SSB, WBFM, CPFSK} 6. {AM_SSB, WBFM, AM_DSB} 7. {WBFM, PAM4, CPFSK} 8. {WBFM, PAM4, AM_DSB} 9. {PAM4, CPFSK, AM_DSB}	n=2916 p=2916 a=3 k=9 r=9 d=6 Per run samples per anomaly n/r/a=108 anomaly samples per dB n/r/a/d=18 samples per known p/r/k=36 known samples per dB p/r/k/d=6	n=5346 p=5346 a=3 k=9 r=9 d=11 Per run samples per anomaly n/r/a=198 anomaly samples per dB n/r/a/d=18 samples per known p/r/k=66 known samples per dB p/r/k/d=6	-
4	**Run# Anomaly Configuration** 1. {GFSK, AM_SSB, WBFM, PAM4} 2. {GFSK, AM_SSB, WBFM, CPFSK} 3. {GFSK, AM_SSB, WBFM, AM_DSB} 4. {AM_SSB, WBFM, PAM4, CPFSK} 5. {AM_SSB, WBFM, PAM4, AM_DSB} 6. {WBFM, PAM4, CPFSK, AM_DSB} 7. {GFSK, AM_SSB, WBFM, PAM4}	n=3888 p=3888 a=4 k=9 r=12 d=6 Per run samples per anomaly n/r/a=81 anomaly samples per dB n/r/a/d=13	n=7128 p=7128 a=4 k=9 r=12 d=11 Per run / samples per anomaly n/r/a=148 anomaly samples per dB n/r/a/d=13	-

	8. {GFSK, AM_SSB, WBFM, CPFSK} 9. {GFSK, AM_SSB, WBFM, AM_DSB} 10. {AM_SSB, WBFM, PAM4, CPFSK} 11. {AM_SSB, WBFM, PAM4, AM_DSB} 12. {WBFM, PAM4, CPFSK, AM_DSB}	samples per known p/r/k=36 known samples per dB p/r/k/d=6	samples per known p/r/k=66 known samples per dB p/r/k/d=6	
5	**Run# Anomaly Configuration** 1. {GFSK, AM_SSB, WBFM, PAM4, CPFSK} 2. {GFSK, AM_SSB, WBFM, PAM4, AM_DSB} 3. {GFSK, WBFM, PAM4, CPFSK, AM_DSB} 4. {AM_SSB, WBFM, PAM4, CPFSK, AM_DSB} 5. {GFSK, AM_SSB, WBFM, PAM4, CPFSK} 6. {GFSK, AM_SSB, WBFM, PAM4, AM_DSB} 7. {GFSK, WBFM, PAM4, CPFSK, AM_DSB} 8. {AM_SSB, WBFM, PAM4, CPFSK, AM_DSB}	n=2640 p=2640 a=5 k=9 r=8 d=6 Per run samples per anomaly n/r/a=66 anomaly samples per dB n/r/a/d=11 samples per known p/r/k=36 known samples per dB p/r/k/d=6	n=4840 p=4840 a=5 k=9 r=8 d=11 Per run samples per anomaly n/r/a=121 anomaly samples per dB n/r/a/d=11 samples per known p/r/k=67 known samples per dB p/r/k/d=6	-*There is a repeat run group, however each group of anomalies are randomly introduced to the classifiers, therefore allowing for variation in the ordering of the anomaly stimulus.
6	**Run# Anomaly Configuration** 1. {GFSK, AM_SSB, WBFM, PAM4, CPFSK, AM_DSB} 2. {AM_DSB, GFSK, AM_SSB, WBFM, PAM4, CPFSK} 3. {WBFM, PAM4, CPFSK, AM_DSB, GFSK, AM_SSB} 4. {PAM4, CPFSK, AM_DSB, GFSK, AM_SSB, WBFM} 5. {CPFSK, AM_DSB, GFSK, AM_SSB, WBFM, PAM4} 6. {AM_DSB, GFSK, AM_SSB, WBFM, PAM4, CPFSK} 7. {GFSK, AM_SSB, WBFM, PAM4, CPFSK, AM_DSB} 8. {AM_DSB, GFSK, AM_SSB, WBFM, PAM4, CPFSK}	n=3888 p=3888 a=6 k=9 r=12 d=6 Per run samples per anomaly n/r/a=54 anomaly samples per dB n/r/a/d=6 samples per known p/r/k=36 known samples per dB p/r/k/d=6	n=7128 p=7128 a=6 k=9 r=12 d=11 Per run samples per anomaly n/r/a=99 anomaly samples per dB n/r/a/d=9 samples per known p/r/k=66 known samples per dB p/r/k/d=6	Mixing up anomaly order to get average stats. Also, the code shuffler randomizes presentation of the data even though these patterns are repeated.

	9. {WBFM, PAM4, CPFSK, AM_DSB, GFSK, AM_SSB} 10. {PAM4, CPFSK, AM_DSB, GFSK, AM_SSB, WBFM} 11. {CPFSK, AM_DSB, GFSK, AM_SSB, WBFM, PAM4} 12. {AM_DSB, GFSK, AM_SSB, WBFM, PAM4, CPFSK}			