INTRUSION DETECTION FOR INTERNET OF THINGS USING DEEP LEARNING APPROACHES

Prepared By

Kiran Kumar Mohanraj - 20054954
MSc Cyber Security
University of the West of England, Bristol

This study was completed as the Dissertation for the MSc in Cyber Security at the University of the West of England, Bristol. This is my work, where the works of others were used or drawn upon, it has been attributed.

INTRUSION DETECTION FOR INTERNET OF THINGS USING DEEP LEARNING APPROACHES

Master of Science

Cyber Security

UWE, Bristol, September 2022

Kiran Kumar Mohanraj

ABSTRACT

Computer networks have rapidly grown as a result of the Internet of Things (IoT) revolution. Even though Internet of Things Devices (IoTDs) improve many aspects of life, they also pose security dangers due to their flaws, which provide hackers with a vast array of potential new targets. To address these issues, IoT applications and networks require lightweight, quick, and adaptable security solutions. In this context, systems built on artificial intelligence and utilising Big Data analytics can show promise in the area of cybersecurity. In this study, Deep Residual Convolutional Neural Network (DRCNN) model, was proposed to detect intrusions and to classify nine attack types present in the TON_IoT dataset. The research was carried out by employing the TON_IoT network dataset which contains network records of various IIoT/IoT devices. This study also proposed Convolutional Neural Network and Long Short-Term memory (CNN-LSTM) model to evaluate and compare both the deep learning approaches. The experimental results showed that the Deep Residual CNN model performed better in classifying the attack types compared to the CNN-LSTM model. Overall, the proposed DL methods performed better than the other IoT based IDS in the literature with the accuracy of 99.77% for the DRCNN model and 99.64% for the CNN-LSTM model.

KEYWORDS: convolutional neural networks, deep learning, distributed denial of service attacks, IoT security, long short-term memory, concurrent neural networks.

ACKNOWLEDGEMENTS

The completion of this thesis would not have been possible without the guidance of my supervisor, Mr. Ian Johnson. Without the thoughtful questions which provided direction for my curiosity, the challenge to push myself toward my goals, and the limitless supply of patience he afforded me I never could have made it this far.

I would also like to thank Dr. Phil Legg, and the other faculty and staff of the FET - Computer Science and Creative Technologies department who provided guidance in the completion of this degree.

Lastly, I would like to thank my parents, relatives, and friends for always supporting and believing in me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Background

The Internet of Things (IoT) is the interconnection of various physical objects like home appliances, vehicles, devices, and other items through public or private network with the help of embedded devices having sensors. The term ‘‘Internet of Things’’ (IoT) was first coined by Kevin Ashton in 1999. He referred the IoT as uniquely identified objects that are connected using radio-frequency identification (RFID) (Ashton, 2009). The IoT has come a long way in its usage and functionality. The IoT devices have become an integral part in minimizing human efforts in many life aspects. The number of IoT devices used worldwide is estimated to become 64 billion by 2025 which is also has the potential to generate 4 trillion to 11 trillion dollars in economic value (Petrov, 2022). Although, the need for securing these devices has always been a challenge for the developers.

The major challenge faced by researchers and security specialists while implementing an IoT environment are security and privacy problems. Lack of regular updates, weak password protection, lack of user awareness and poor IoT device management are some of the challenges faced by the IoT (Thales Group, 2021). For various business enterprises and public organisations, the security and privacy concerns in IoT have significant consequences. For example, the advantages of IoT in healthcare sector have contributed to people's improvement in perceptions of their health. However, as the number of devices increases, the advantages come with noticeable concerns. The IoT ecosystem's expanding connected device population could pose security challenges because it gives hackers and thieves more access points (Tawalbeh *et al.*, 2020).

To overcome these challenges, various security protocols and systems are implemented. An Intrusion Detection System (IDS) is one such solution which operates in the IoT system's network layer to look for any unwanted requests and intrusions in the IoT network. A deployed IDS for an IoT system should be able to analyse data packets and produce responses in real time, analyse data packets at various IoT network layers using various protocol stacks, and adapt to various IoT technologies (Elrawy, Awad and Hamed, 2018). Machine learning has been demonstrated to be extremely accurate at detecting intrusion in the network, although it is computationally expensive (Diro and Chilamkurti, 2018). While highly accurate, deep learning algorithms frequently require the usage of GPU acceleration to ensure low runtime latency. These methods have achieved state-of-the-art results across numerous application domains. Unfortunately, the cost of GPUs designed for artificial intelligence puts them out of the reach of the typical user (Goodfellow, Bengio and Courville, 2016).

1.2 Research Motivations

Implementing an Intrusion Detection System (IDS) for IoT network has always been a challenge for researchers. The research works on IoT based IDS mostly focuses on either implementing low energy consumption devices with few security measures or implementing intelligent intrusion detection techniques while facing significant false positive rates (da Costa *et al.*, 2019). Also, most of the datasets used for these studies are outdated and are not specific to IoT networks (Booij *et al.*, 2022). This research work is highly motivated to develop an IDS based on deep learning approaches which performs effectively and efficiently with high detection rates.

1.3 Research Contribution

The key contributions of this research work are presented below.

- A deep learning algorithm named Deep Residual Convolutional Neural Network (DRCNN) was proposed to detect intrusions and to classify the attack types.
- Along with the DRCNN model, the CNN-LSTM model was also proposed which were utilized to classify the attacks in the TON_IoT dataset.
- Both the models were evaluated using the evaluation metrics which showed high detection accuracy with less resource consumption and time.

1.4 Ethical Considerations

The research work did not involve any ethical issues as it only deals with the dataset published by (Alsaedi, Moustafa, Tari, Mahmood and Anwar, 2020) which is publicly available for researchers to work on it. The research does not involve any participants which rules out the ethical considerations for anonymity, confidentiality and potential for harm.

1.5 Overview

The research is organized as follows: the review of various types of IDS in IoT were explored with importance on ML and DL based IDS were described in Chapter 2, followedby the literature review of IoT based dataset were discussed in Chapter 3 with specific focus on TON_IoT dataset. Chapter 4 presents the research method with the research approach for carrying out this thesis. It also presents the novel deep learning approaches for detecting and classifying network attacks. Chapter 5 contains the results and discussion of the thesis. Concluding remarks and directions for future work are discussed in Chapter 6.

2 LITERATURE REVIEW

In this section, we will be looking into the detailed review of intrusion detection systems in IoT followed by an amalgamation of machine learning, and deep learning techniques for detection.

2.1 Intrusion Detection System in IoT

The IDS is used to observe the unauthorized activity in network traffic. The goal of IoT IDS is to monitor the network traffic and end nodes, detect any malicious activity or irregular patterns in the network, and inform the end user about the intrusion (Sherasiya, Upadhyay and Patel, 2016). In the IoT ecosystem, the compromise of the availability, integrity, or confidentiality of IoT systems are considered as malicious actions which are detected by the IDS. IDS system has two main sub-categories: Signature-based Intrusion Detection System (SIDS) and Anomaly-based Intrusion Detection System (AIDS).

2.1.1 Signature-based Intrusion Detection System (SIDS): It is also known as Knowledge-based Detection or Misuse Detection. SIDS detects any intrusions by analyzing the network packets for any specific patterns. The SIDS maintains a database for signatures of the well-known attacks which are used to compare the signature of incoming network traffic for intrusions (Panagiotou *et al.*, 2021). (Kreibich and Crowcroft, 2004) proposed a honeypot system which produces IDS signatures by analyzing the network traffic on the honeypot. The system was able to detect the signatures of previously known attacks accurately. (Zeidanloo *et al.*, 2010) noted that SIDS provides immediate detection to intrusion with no false-positives form known attacks. This helps the users to provide swift response to contain the infected system and avoid further damage to the network.

SIDS have been implemented in various tools, such as Snort (Roesch, 1999) and Suricata. These tools are compared by several authors to analyse the performance of SIDS. (Waagsnes and Ulltveit-Moe, 2018) found significant differences in the ability to detect any intrusions by Snort and Suricata in an Industrial Control Systems (ICS) networks. Contrastingly, these differences were not noticed by (Albin and Rowe, 2012), the authors were able to find only the difference in computational costs when the tools were used as detectors in real IT scenario.

(Douligeris and Mitrokotsa, 2003) notes that there is significant increase in detection time when the signature of attacks in the database increases. This is because the databases need to update regularly for new attacks to be more effective. Another major disadvantage of SIDS is its ineffectiveness to detect any variant of known attacks, zero-day attacks or any new attacks. (Hoang and Nguyen, 2018) renders SIDS as ineffective, present-day attacks are more advanced in making itself stealth during code analysis and are updated with dodging techniques. Due to these disadvantages of being static, anomaly-based intrusion detection system (AIDS) are widely implemented.

2.1.2 Anomaly-based Intrusion Detection System (AIDS): The AIDS works based on detecting anomaly in the network behavior. The system models the normal behavior of the network, an alarm is generated when there is a difference in the incoming traffic's behavior with the normal traffic. Since the IDS can detect the signatures of the novel attack as intrusion because of its abnormality from usual behavior, the anomaly-based detection is preferred more than SIDS. The AIDS works better when the rules and protocols of the network are well defined for the system. Various vendors define their own protocols which makes it difficult

for the network administrators to define rules for the IDS. When the rules are not defined properly, the malicious behavior in the network is considered as normal behavior, thus allowing attackers to go unrecognized (Jyothsna, Prasad and Prasad, 2011).

The Anomaly-based detection techniques can be divided into four categories (Gyanchandani, Rana and Yadav, 2012): Statistical anomaly detection, Data mining-based detection, Knowledge based detection, and Machine learning based detection. Statistical based anomaly detection techniques make use of statistical properties to model the normal behavior and tests the incoming traffic with the normal profile to detect deviation. Statistical methods provide better results for attacks like denial-of-service (DoS). The Data mining-based detection techniques creates more meaningful models to even detect insider attacks which is difficult for other IDS to detect. The process involves looking for useful pattern from large amounts of network data to create network models. Knowledge based detection techniques is like the SIDS where the system gains knowledge about the various attacks and vulnerabilities, this information is used to generate an alarm when there is an intrusion. This technique is effective when the knowledge of the vulnerabilities and attacks are updated regularly. Machine learning based detection techniques is based on updating itself using its previous results to improve the performance of the system (Gyanchandani, Rana and Yadav, 2012). We will be looking in detail about Machine Learning (ML) based IDS in the following section.

Various author (Feily, Shahrestani and Ramadass, 2009; Zeidanloo *et al.*, 2010; Amini, Araghizadeh and Azmi, 2015) have stated that the major drawback of AIDS is its difficulty to define normal behavior of the network and to set a threshold for detecting intrusions because of the changes in network activity over time. Due to these reasons, hybrid systems of

Anomaly and Signature-based implementation are widely used in insecure networks.

2.2 Machine Learning based IDS

The Machine Learning is a computational algorithm, which is a branch of Artificial Intelligence (AI). It is designed to imitate the human intelligence by understanding and building models to learn from the data provided (el Naqa Issam and Murphy, 2015). The use of machine learning techniques for implementing an IDS has been increasing over the years. These techniques help in finding patterns in network traffic data and use these models to find deviations. The ML techniques are broadly classified as supervised and unsupervised (Khraisat and Alazab, 2021).

2.2.1 Supervised learning: Supervised learning-based techniques uses the labelled data for training itself to detect any variations. Firstly, the labelled training data is provided with the classified features, the model learns and improves its performance with the provided labels. Finally, the testing data is used to test the model's effectiveness. There are various methods of implementing supervised learning techniques, namely, Decision trees, Naïve Bayes, Genetic Algorithm (GA), neural networks, support vector machines, and k-nearest neighbour (Khraisat and Alazab, 2021).

2.2.1.1 Decision Trees: A Decision Tree is tree-like graph with various internal decision nodes, leaf nodes and branches. The internal nodes denote a test attribute, while the leaf node represents a class label. The branches signify the outcome of the test attribute (Rai, Devi and Guleria, 2016). (Sivatha Sindhu, Geetha and Kannan, 2012) constructed a lightweight IDS based on Decision tree algorithm. The authors proposed a hybrid algorithm

combining neural network and decision tree which helps in increasing the generalization ability and the ability to classify unseen attack pattern. The proposed neurotree classification algorithm was able to detect attacks with 98.4% accuracy and was more suitable for multi-class classification. (Rai, Devi and Guleria, 2016) proposed a Decision Tree Split (DTS) algorithm based on C4.5 decision tree approach which address the problems of constructing decision tree. The authors constructed this algorithm by choosing important features of the data using information gain and selecting the split value of the tree which makes unbiased classification. The proposed algorithm was able to detect intrusion more efficiency with less features when compared with other classifiers.

2.2.1.2 Naïve Bayes: The Naïve Bayes technique is based on Bayes' principle where the classification is done according to probability of an attack occurring with the normal behavior. Although this method is widely used, the Naïve Bayesian network is restricted because of its independence between information node (Panda and Ranjan Patra, 2007). The authors (Panda and Ranjan Patra, 2007) proposed a framework of IDS based on Naïve Bayes classifier algorithm. Although the algorithm achieved high accuracy with less time consumption, the number of false positives produced was quite high. (ben Amor, Benferhat and Elouedi, 2004) compared the performance of the Naïve Bayes networks with the decision tree using KDD'99 intrusion datasets. Though the decision tree provided slightly better accuracy in detecting the attacks, the construction of naïve bayes is largely faster than the decision tree. To overcome the flaws of naïve bayes method, which considers features are independent of each other, (Koc, Mazzuchi and Sarkani, 2012) introduced Hidden Naïve Bayes multiclass classifier. This model augmented with various discretization and feature selection methods produced better accuracy in attack detection and less error rate, which

makes it better model for datasets with dependent attributes.

2.2.1.3 Genetic algorithms (GA): GA is method of applying biological evolution theory to computer systems. It denotes each possible solution as chromosomes and uses set of genetic operators to improve the quality of the results over the time. In (Chittur, 2001), the author was able to successfully implement a GA based model for intrusion detection with high detection rate and low false positives. Through random mutation, GA was able to evolve the detection model to detect intrusions from the data not seen before. There has been a hybrid implementation of GA with other algorithms like Fuzzy (Haghighat *et al.*, 2007) and support vector machine (Kim, Nguyen and Park, 2005) in order to improve the overall performance of the IDS. In (Haghighat *et al.*, 2007), the author used GA as search algorithm along with other evolutionary algorithm which provides efficient solutions for complex optimization problems by patterning from natural treatments. Whereas (Kim, Nguyen and Park, 2005) fused the GA with the SVM which was able to select optimal feature set from the whole feature set. This improved the performance by reducing the number of features that SVM should process and increasing the detection rates.

2.2.1.4 Artificial neural network (ANN): ANN is based on the working of the nervous system of the human brain. The artificial neurons are the elementary processing units of the network. The network contains an input layer, multiple hidden layers, and an output layer. The ANN learning technique is based on the backpropagation algorithm (Ahmad *et al.*, 2020). The backpropagation algorithm adjusts its weights to rectify the network error in each iteration. (Ryan, Lin and Miikkulainen, 1997) published one of the first works of implementing Neural network for intrusion detection. The 2-layer Multilayer-perceptron (MLP) architecture was

used for their system of ten users. For training the model, the backpropagation algorithm was used. Their system was evaluated based on the false positives and false negative rate for detecting intrusion which were 7% and 4% respectively. In 2012, (Shah and Trivedi, 2012) provided a survey on various implementation of ANN based approach for anomaly detection. After comparing various approaches, the author states that approaches using simple ANN are fast but compromises in accuracy, whereas the multilayer ANN approach consumes more time but are accurate in detecting intrusions. There are many variants of ANN which will be discussed in the later section of this report.

2.2.1.5 Support vector machines (SVM): SVM is a discriminative classifier which predicts the target values of the testing data based on it attributes by a splitting hyperplane. Though SVM is memory efficient, it struggles to perform on a noisy data which contains overlapping classes (Bhati and Rai, 2020). The SVM was introduced by (Boser, Guyon and Vapnik, 1992) as a classification and regression tool for making some predictions in COLT-92. In the research works of (Fischetti, 2016), the author introduced a new model of SVM which uses Gaussian kernel function to find out the similarities. The Mixed-Integer Linear Programming model (MILP) was implemented to reduce overfitting on the training dataset. The results showed slight improvement in the training time and better classification accuracy. (Horng *et al.*, 2011) combined hierarchical clustering algorithm and the SVM technique, to propose an IDS. The Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) hierarchical clustering features provided SVM with higher-qualified training instances. The IDS provided better results in detecting DoS and Probe attacks and good overall accuracy.

2.2.1.6 Fuzzy logic: The fuzzy logic was first introduced by Lotfi Zadeh in 1965 which is a technique based on the degrees of uncertainty. This technique is good for implementing as IDS classifier because of its mathematical representation of vagueness and imprecise information. There is various implementation of fuzzy logic-based technique for IDS, namely, fuzzy clustering method, fuzzy classifiers, fuzzy feature extraction, and so on (Masdari and Khezri, 2020). The two most widely used misuse detection method based on fuzzy logic are ANFIS or adaptive neuro-fuzzy inference system and Fuzzy clustering. The ANFIS combines the learning feature of the ANNs with the uncertainty handling capability of the fuzzy logic. In (Panja, Ogunyanwo and Meharia, 2014), the researchers proposed a scheme that combined ANFIS classifier and fuzzy rules for detecting intrusions in the network. For data classification, the GA was used with the ANFIS. The IDS was able to reduce the false positive rate and improved the accuracy. In the research work of (Wang *et al.*, 2010), the authors introduced an IDS scheme combining fuzzy clustering and ANN model. The fuzzy clustering technique was used to generate different training subsets which was then used to train the ANN model. The experimental results were able to produce better detection rate and less false positive rate than other well-known methods like decision tree and naïve bayes.

2.2.1.7 Ensemble methods: The ensemble methods are based on ensemble learner which is made up of different classifiers. The base learner is trained and provides classification of a particular class. All the predictions of these base learners are collaborated using combiner. This technique performs better than an individual classifier in areas such as incremental learning, data fusion, feature selection and error correcting output codes (Polikar, 2006). The early works on ensemble learning by (Dietterich, 2000), showed that the ensembles works better than any single classifier. The researcher reviewed algorithms like

Bayesian averaging, error-correcting output coding, Bagging, and boosting to compare its performance with other single classifiers. According to the systematic study on ensemble learning for IDS by (Tama and Lim, 2021), the random forest classifier was widely implemented for IDS because of its diverse nature and is easy to use. The authors stated that in most of the cases the ensemble learning has provided significant improvements compared to the individual classifiers.

2.2.2 Unsupervised learning: It is a type of ML technique which detects patterns from the training dataset without using class labels. Through the process of learning the input data, the model groups the data into various classes. The incoming traffic is classified according to its similarities to a particular group. The most common unsupervised learning technique is the K-means technique. It is a type of clustering algorithm which finds the arithmetic mean of all the data objects of a cluster. Then, it allocates every data object to the nearest cluster. (Jianliang, Haikun and Ling, 2009) presented IDS based on K-means algorithm. The experiment was conducted on KDD-99 dataset which showed that the algorithm was effective in partitioning large dataset. The author stated that the K-means algorithm has better global search ability. In (Peng, Leung and Huang, 2018), the authors proposed a clustering method based on Mini Batch K-means with PCA (PMBKM) for Intrusion Detection System. The experimental results showed that the PMBKM was more efficient and effective than the other algorithms such as K-means and K-means with PCA (PKM).

2.3 Deep Learning-based IDS

Deep learning (DL) is the subset of machine learning where the system uses multiple hidden layers to understand the characteristics of deep network. Similar to ML techniques, deep learning is classified as supervised and unsupervised. The DL approaches contain multiple neural nodes and one classifier known as neural unit or perception. The hidden layers are present between the input and output and these nodes are called as hidden nodes. The major drawback of ML methods is that it involves complex hypothesis to be written for the system to do prediction. Whereas in DL methods, the neutral network generates these hypotheses on our behalf. The ML approach requires small vectors of feature selection for classification but in DL approach the raw data could be used rather than selecting important features and throwing away most of the data (Dong, Wang and Abbas, 2021). This section presents a review of IDS based on deep learning approaches in the security literature.

2.3.1 Autoencoder (AE): AE is type of unsupervised neural network which produces output close to input by understanding the features of the dataset. The AE contains an encoder and a decoder, where the encoder compresses the input data into bottleneck referred to as the latent space variables. The decoder reconstructs the original data from these latent space variables (Finke *et al.*, 2021). (Farahnakian and Heikkonen, 2018) proposed a deep auto-encoder based IDS (DAE-IDS). This system consists of four-layer auto-encoder connected in a sequential manner. The DAE-IDS detects anomalies through unsupervised feature learning and supervised fine-tuning. To evaluate the performance, the system was experimented on the KDDCup99 and achieved 94.71% accuracy in detecting intrusions. (Yan and Han, 2018) put forward a deep learning-based IDS using stacked sparse autoencoder (SSAE) and SVM. The proposed SSAE with optimal structure nonlinearly maps high-dimensional features to low-

dimensional feature without losing much of the information from the original data. The SSAE was able to perform better than other ML classifier with high detection rates and with less resource consumption. The major drawback of this method is that it failed to detect R2L and U2R low-frequency attack samples. In the research works of (Andresini *et al.*, 2020), the authors presented Multi-channel Deep Feature Learning for intrusion detection. Two auto-encoders were trained unsupervised to learn from normal and attack flows to reconstruct the samples again. These autoencoder feature vectors and the original feature vector representation of network flow are provided to one dimensional Convolution Neural Network (CNN) model which then classifies the dataset. The proposed method showed superior performance compared to other DL methods but failed in classifying minority classes. This IDS do not provide any information about the characteristics of the attacks.

2.3.2 Recurrent neural network (RNN): The RNN is designed to model a series of data with variable length which can work efficiently. Like any other DL technique, the structure of RNN consist of input, hidden and output units, where the hidden units act as memory elements. The RNN makes a decision by using the information of its previous state as input to its current state and repeats this process for a particular number of steps. This process of RNN makes it effective for working with sequence of data which repeats over time. This technique provides RNN a short-term memory (Ahmad *et al.*, 2020). (Yin *et al.*, 2017) introduced RNN based IDS for multiclass and binary classification of attacks in NSL-KDD dataset. The model was analyzed different hidden nodes and various learning rate's impact on detecting the intrusions. The experimental results showed that the best accuracy and performance was recorded when using 80 hidden nodes and with the learning rate of 0.1 for binary classification and 0.5 for multiclass classification. The proposed model performed

better when compared with various ML techniques like naïve Bayesian, J48 and random forest. The major drawback of this model is its low detection rate for the R2L and U2R classes. In the works of (Xu *et al.*, 2018), the authors proposed RNN based IDS with gated recurrent units (GRU) as the primary memory unit along with softmax module and multilayer perceptron (MLP). The proposed model with GRU as memory element and MLP for network intrusion showed better detection rates than the LSTM with RNNs and bidirectional GRU, when tested using KDD Cup'99 and NSL-KDD datasets. (Jiang *et al.*, 2020) introduced LSTM based RNN technique for detecting intrusions and to do multiclass classification of attacks. The authors improved the detection rate by preprocessing the data and performing feature abstraction and training. The proposed model outperformed classifies like Bayesian and SVM.

2.3.3 Deep neural network (DNN): The DNN is similar to the other DL structures with an input layer, an output layer, and multiple hidden layers which allows the model to learn and predict intrusions. The DNN technique is used for problems involving complex nonlinear functions. The use of multiple hidden layers improves the abstraction level of the model to increase its capability (Ahmad *et al.*, 2020). (Potluri and Diedrich, 2016) introduced DNN based IDS scheme which mainly focuses on discovering complex relations between the input dataset for detecting intrusions. The input data is processed, and the training is conducted on various multi-core systems. The experimental results showed the system was efficient and effective in detecting attacks like DoS and Probe when tested with the NSL-KDD dataset. The system was not effective in detecting attacks like R2L and U2R because of its insufficient data. The authors stated that the results showed the acceleration of training process by using the multi-core CPUs was faster than training in a serial manner. Similarly, (Jia, Wang and Wang, 2019) proposed a new deep neural network (NDNN) based IDS with four hidden

layer to classify KDD cup'99 and NSL-KDD datasets. Although the proposed model was able to perform better than the traditional ML techniques, there were some problems that needs to be addressed. Since there were too many parameters used, it is difficult to tune these when the structure of the network increases. The optimization algorithms and automatic tuning techniques are required to enhance the performance of the model.

2.3.4 Deep belief network (DBN): The DBN is type od DL method which is made up of multiple layers of Restricted Boltzmann Machines (RBM) followed by a softmax classifier. A RBM is a two-layer model with binary stochastic hidden units and binary stochastic visible units in each layer. The structure of DBN is that no connection is made between nodes of same layer rather these nodes are connected to previous and next layers (Mohamed, Dahl and Hinton, 2009). (Gao *et al.*, 2014) in their research proposed DBN based IDS which integrates back-propagation network and RBM. The authors used unsupervised greedy learning algorithm for pre-training and fine tuning the DBN, so that the model learns high-dimensional representation of input and performs the classification tasks. The experimental results showed that the four hidden layer RBM produced better performance when tested on the KDD Cup'99 dataset. (Wei *et al.*, 2019) proposed a model to construct and optimize the DBN based IDS. The researchers used the PSO algorithm as the initial optimization method which was then used as initial partical swarm of GA optimization. This algorithm was then used to optimize the DBN model. The proposed model was able to produce high detection accuracy and speed. However, the model needs to be optimized further to reduce the training time due to its complex nature.

2.3.5 Convolutional neural network (CNN): The CNN is similar to the traditional ANN with neurons that self-optimise through learning, but it consists of convolutional and pooling layers for feature extraction and a fully connected layer and softmax classifier for classification. The CNN method is widely used for recognizing patterns within the images (O'Shea and Nash, 2015). (Xiao *et al.*, 2019) introduced an intrusion detection model based on CNN. The authors performed feature extraction using Principle Component Analysis and AE. For providing the input data to CNN model, the dataset is converted into a two-dimensional (2D) matrix through dimensionality reduction. The proposed model provides better detection rates and significantly reduces the classification time. (Zhang *et al.*, 2019) proposed a IDS model named DCF-IDS which combine CNN with gcForest (deep random forest) to form a multilayer model. The researchers have also proposed a novel P-Zigzag encoding algorithm which converts the raw input traffic data into 2D greyscale images. The model uses two layers of classifiers, first layer which uses GoogLeNetNP for initial detection, then the second layer uses caXGBoost model for fine detection of abnormal classes into N-1 subclasses. The experimental results tested on UNSW-NB15 and CIC-IDS2017 datasets showed significant increase in accuracy and detection rates with very few false alarm rates. In the works of (Yang, Cheng and Chuah, 2019), the authors implemented an IDS for SCADA systems based on CNN model. The proposed CNN model helps in modelling the temporal network behavior patterns of SCADA hosts. The high detection accuracy is achieved by applying re-training schemes for unknown attacks. However, the model needs to be developed further to support mixed attacks and enhance itself when dealing with attacks on other SCADA protocols.

3 TON_IOT DATASET

With the increase in the number of network attacks recently, the need for implementing an automated intrusion detection system has risen. Security researchers have developed various IDS using ML and DL techniques to detect intrusions in a network. But most of these models are tested using generic network traffic dataset namely NSL-KDD (Tavallaee *et al.*, 2009) and KDD CUP'99 ('KDD dataset', 1999). Only few datasets have been publicly available which is specifically for detecting intrusions in IoT network. In this section, we will be looking into these IoT network intrusion datasets and about the TON_IoT dataset which is used for this research.

3.1 Review of IoT dataset for Network IDS

The IoT datasets taken into consideration for this review are UNSW-IoT (Sivanathan *et al.*, 2019), BoT-IoT dataset (Koroniotis *et al.*, 2019), N-BaIoT dataset (Meidan *et al.*, 2018), IoT Network Intrusion dataset (Kang *et al.*, 2019) and Aposemat IoT-23 dataset ('Aposemat IoT-23', 2022).

3.1.1 UNSW-IoT Trace dataset: This dataset was introduced by the researchers from University of New South Wales (UNSW), Sydney, who instrumented a smart environment with 28 unique IoT devices to collect IoT network traffic continuously over 26 weeks. The devices used for emulating a smart environment are cameras, lights, plugs, motion sensors, appliances, and health-monitors. The statistical attributes used for getting insights about the network characteristics are activity cycles, port numbers, signaling patterns and cipher suites. The authors also developed a ML based classification framework which produced over 99 percent accuracy in identifying the IoT devices. There were some flaws in the model where

the short traffic traces and traces of attack traffic were ignored from consideration because of the partition of devices by its MAC address (Sivanathan *et al.*, 2019).

3.1.2 BoT-IoT dataset: The BoT-IoT dataset was developed by (Koroniotis *et al.*, 2019) in Cyber Range Lab of UNSW Canberra Cyber, Australia, which was specifically designed for identifying Botnets in IoT networks. The dataset was developed on realistic testbed which consist of devices like IoT sensors, including a weather station, a smart fridge, motion activated lights, a garage door, and a thermostat. The dataset is classified into normal network traffic, probing attacks, DoS and DDoS attacks, and Information Theft attacks. Since the dataset collects data using simulated IoT devices, it fails in representing the real IoT traffic. (Peterson, Leevy and Khoshgoftaar, 2021) analysed the Bot-IoT dataset and stated that there were several invalid features which were considered by various researcher for their studies. The BoT-IoT dataset needs to be adopting an effective data cleaning procedure.

3.1.3 N-BaIoT dataset: The N-BaIoT dataset introduced by (Meidan *et al.*, 2018) contains statistics of network traffic of normal behaviour and attack cases collected from nine IoT devices. This dataset is specifically designed for detecting network-based botnets and the attacks considered are BASHLITE and Mirai botnet. The network characteristics like host-IP, host-MAC&IP, channel, network-jitter, and socket are aggregated to form five major feature categories. The dataset contains 23 statistical features for these five feature categories extracted from the five-time windows. The authors implemented an anomaly detection technique using deep autoencoders which produced high detecting accuracy (Meidan *et al.*, 2018).

3.1.4 IoT Network Intrusion dataset: The researchers from the Hacking and Countermeasure Research Lab introduced this publicly available dataset (Kang *et al.*, 2019) which is designed to simulate an IoT environment. The dataset was developed using two smart home devices which are SKT NUGU Model NU 100 AI Speaker and EZVIS Wi-Fi Camera Model C2C Mini O Plus along with some smart phones and laptops. These devices were connected to same WiFi network to simulate the normal traffic and 4 different attacks scenarios, namely Scanning attacks, Man in the Middle (MITM) attacks, Denial of Service (DoS) attacks, and Mirai Botnet attacks, using Nmap tool. The dataset contains 42 raw network packet files collected at different time intervals. Since only two IoT devices were used to collect data, the dataset is not heterogenous in nature which is very important for identifying diverse network attacks (Booij *et al.*, 2022).

3.1.5 Aposemat IoT-23: The most recently published labelled IoT dataset by Stratosphere Laboratory, Avast AIC in association with Czech Technical University (CTU), Czech Republic is the Aposemat IoT-23 dataset which is available at ('Aposemat IoT-23', 2022). The dataset was developed by capturing data from IoT devices like a Philips HUE smart LED lamp, an Amazon Echo home intelligent personal assistant, and a Somfy Smart Door Lock for a period from 2018 to 2019. There are 23 attack types of which three are normal traffic behaviour of IoT devices and other 20 are malware traffic after compromising the devices by malwares like Mirai, Torii, Trojan, Gagfyt, Kenjiro, Okiru, Hakai, IRCBot, Muhstik, and Hide & Seek. The network traffics of the IoT environment are categorised into attack traffic, benign traffic, C&C traffic, and file download traffic.

3.2 TON_IoT Dataset Description

The TON_IoT dataset is considered for this research work. This dataset includes heterogeneous data sources collected from Industrial IoT (IIoT) ecosystem. It contains telemetry data from IIoT and IoT sensors, Windows and Linux operating system logs, and network traffic of IIoT systems. The dataset was generated at the Cyber Range and IoT Labs at the UNSW Canberra, Australia from a realistic depiction of a medium scale IoT network. The TON_IoT dataset is publicly available and can be accessed at (Alsaedi, Moustafa, Tari, Mahmood and Anwar, 2020).

The testbed environment was built by (Alsaedi, Moustafa, Tari, Mahmood and Adna N Anwar, 2020) for generating the TON_IoT dataset with real-world representation of IoT network data. The layers of the testbed are edge/IoT layer, fog layer and cloud layer. The Edge layer is the endpoint which contains the physical IoT devices and its operating systems. The sensors data and readings of the IIoT/IoT devices are collected in this layer and passed onto the Fog or Cloud layer for analysis. The edge layer consists of IIoT/IoT devices like smart TV, smartphones, sensors like weather and smart light bulbs, servers, and physical gateways. The NSXVMware hypervisor platform was installed to manage the VMs present at the fog layer. The Fog layer consists of virtualization technology which helps in controlling the VMs and their services using the NSXVMware hypervisor platform. The fog layer helps in providing some analytical capabilities of the cloud layer to provide limited capacity and storage near the edge layer with data sources. The Cloud layer provides cloud services to analyze and visualize the behaviours of the edge devices. The HIVE MQTT broker cloud service, PHP vulnerable website and cloud center services like Microsoft Azure or Amazon Web services were used for further analysing the IoT data (Alsaedi, Moustafa, Tari, Mahmood

and Adna N Anwar, 2020).

The TON_IoT dataset has more than 450k network records of normal behaviour and attack scenarios with the labels of its sub-categories. There are 44 feature which were extracted by simulating the IoT network. The complete list of all the features with its description are provided in Appendix II. Figure 1 and 2 shows the statistics of the dataset with 300k records of benign behaviour while 1,61,043 records of malware. (Alsaedi, Moustafa, Tari, Mahmood and Adna N Anwar, 2020) simulated various attacks for generating the TON_IoT dataset. The nine attack families used in this dataset are:

a) *Scanning attack:* It is a method of scanning the network for vulnerable devices to gather information and execute any type of attack. The Nessus and Nmap tools are used to execute the scanning attacks.
b) *Denial of Service (DoS) attack:* The attacker launches malicious request to flood the victim's system to disrupt the access to other legitimate users. The Scapy package was used for the Python scripts to create the DoS attack scenarios.
c) *Distributed DoS attack:* It is similar to the DoS attack but the only difference is that the malicious request is sent by multiple compromised devices called botnets to disrupt the service. The Python scripts and ufonet toolkit was utilized for executing the attack.
d) *Ransomware Attack*: It is a type of attack where the attacker encrypts the victim's system and sells the decryption keys to the victim to gain access back to the system. The Metasploit tool was used to implement the attack against the systems running Windows OS.
e) *Backdoor Attack*: In this attack, the attacker gains unauthorized access to the victim's system through bypassing all the security measures with the help of a backdoor

malware. The Metasploitabe3 framework was used for implementing the attack.

f) *Injection Attack*: The adversary injects a malicious code or query or data to execute it at the backend to gain unauthorized remote access to the system. For simulating these attacks, two bash scripts were injected into input data fields of the vulnerable public PHP and DVWA web applications.

g) *Cross-Site Scripting (XSS)*: It is a vulnerability which allows the adversaries to inject malicious scripts into web applications, thus allowing the attacker to access the data of the user who tries to reach these websites. The malicious bash scripts of Python code using XSSer toolkit was utilized for executing the attack on the vulnerable websites.

h) *Password cracking attack*: The attacker uses brute-forcing techniques or dictionary attacks to guess the password of the victim's system to gain unauthorised access. The Hydra and Cewl toolkits were used for brute-forcing the password.

i) *Man-in-the-Middle (MITM) attack*: It is a type of attack where the actors intercepts the communication between two users and manipulates the shared data. For simulating this attack, Ettercap tool was used to execute ARP Cache poisoning, ICMP redirection and port-stealing attacks.

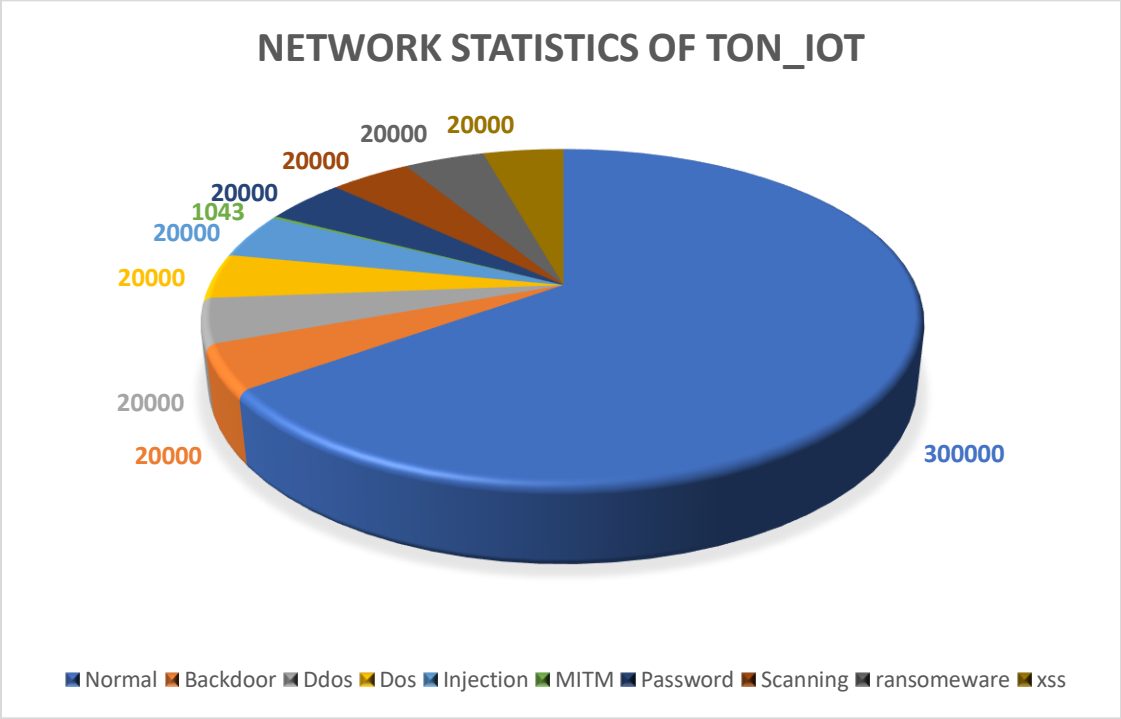


Figure 1 Network statistics of TON_IoT dataset

3.3 Review of IDS based on TON_IoT dataset

The TON_IoT dataset is becoming popular among researchers looking for dataset which is specific to IoT network because of its heterogeneity. (Booij *et al.*, 2022) analysed the TON_IoT dataset with the existing IoT network datasets based on the property of heterogeneity. The authors stated that the TON_IoT dataset depicts the data more closely to the real-world industry IoT network because of its diverse range of attack scenarios and wide varieties of IoT devices used. Most of the existing IoT network datasets contain packet/low-level information or a combination of both to detect attacks on IoT networks but the TON_IoT dataset stands out from others by including the actual data generated from sensor readings. In this section we will be reviewing various IDS implemented using the network records of TON_IoT dataset.

(Khan *et al.*, 2022) proposed a novel IDS specific for smart healthcare system driven

by Internet of Medical Things (IoMT). Their works address the importance of security in IoMT and to build explainable AI for healthcare systems which is critical for increasing the trust in real-life applications. The novel bidirectional simple recurrent units (SRU) model (named XSRU-IoMT) was proposed which uses skip connections to effectively identify attack vectors and solve the fading gradient issue in recurrent networks. The research dives into both the performance factor and interpretability view of an ML/DL model which is effective in identifying the issues in the model and to understand it better. The experimental results shows that the proposed model has high accuracy in detecting the intrusions compared to other state-of-the-art compelling methods.

(Gad, Nashat and Barkat, 2021) introduced IDS for Vehicular Ad Hoc Networks (VANETs) based on the TON_IoT dataset. Different ML techniques were compared for its performance and concluded that XGBoost has the highest detection rates among others. The authors conducted various data preprocessing techniques on the TON_IoT dataset which showed significant increase in the detection rates. The Chi2 technique was used for feature selection while the SMOTE technique was used for class imbalance. These techniques reduced the complexity and overfitting of the model and showed better performance. The IDS was implemented for both binary classification and muti-class classification where the XGBoost techniques produced more than 98% accuracy in both the cases. However, the system requires better optimization algorithm for dimensionality reduction.

(Latif *et al.*, 2021) introduced a novel Dense Random Neural Network (DnRaNN) based IDS for detecting intrusion and classify the attack families. The proposed model contains one input layer, four dense cluster hidden layer and an output layer. The multilayer

architecture was formed by the dense cluster framework which contains numerous nuclei that communicate with each other using synapse and soma-to-soma interactions. The TON_IoT dataset was used to evaluate the performance of the proposed model. The data preprocessing and data normalization techniques were conducted before providing the data to the model. The performance of the model and its detection rates was evaluated for both binary and multiclass scenarios, which produced more than 99% accuracy in detecting intrusions. The proposed model performed with much effectiveness when compared with other ML/DL methods. The drawback of this research method is that the authors failed to preprocess the data effectively by eliminating unnecessary features and replacing the missing values with suitable data.

(Lo *et al.*, 2022) presented a novel Graph Neural Network model named E-GraphSAGE that considers the edge information in the embedding process along with the topological pattern of a network flow graph to detect any intrusions. The authors were the first to introduce GNN based IDS for the IoT network. The proposed model was evaluated using the BoT-IoT and TON_IoT datasets along with the Netflow versions of these datasets. For data preparation the Bro- IDS network monitoring tool was used to generate the network flow features of the TON_IoT dataset. The proposed NIDS based on E-GraphSAGE performed well against the ML classifiers with high detection accuracy. Nevertheless, the runtime of the model needs improvement by applying non-uniform sampling techniques.

(Abdel-Basset *et al.*, 2022) proposed federated deep learning-based intrusion detection framework (FED-IDS) for detecting cyber-attacks in heterogeneous Intelligent Transportation System (ITS). The authors evaluated the model in heterogenous network using TON_IoT dataset and to detect intrusion within in-vehicle communications the Car-Hacking datasets

was used. The proposed framework utilizes the blockchain-secured training and coordination between vehicular edge nodes to improve the efficiency and flexibility of learning on decentralized data sources. The experimental results showed that the proposed framework was very effective and efficient in detecting the intrusions. However, the implemented blockchain system demands high computational power and limited proficiency. The researchers failed to focus on the class imbalance problems in the datasets which affected the performance of the system.

4 RESEARCH METHOD

The literature review showed that the IDS implemented based on ML/DL approaches performs exceptionally well compared to the traditional methods. This research is motivated by the works of (Alsaedi et al, 2020; Gad, Nashat and Barkat, 2021; Latif et al., 2021), to implement DL techniques for detecting network intrusion using the TON_IoT dataset. This research work addresses the flaws discussed in the literature review of IDS based on TON_IoT dataset, by implementing various data preprocessing techniques and proposes the Deep Residual CNN model which performs effective and efficient. This chapter outlines the significance and aim of this project. Then, the research approach is presented which describes the proposed system followed by the experimental methodology and its implementation.

4.1 Research Approach

The proposed system is based on Deep residual 1D CNN model to detect intrusions in the IoT network and to classify the attack types. The research methodology is divided into different phases namely data preprocessing, feature selection, model training and model evaluation. Figure 2 shows the overview of the proposed Deep Residual CNN based IDS. Firstly, the TON_IoT dataset is acquired, and its network data is preprocessed before providing it to the model to achieve high performance. The optimal features are selected from the preprocessed so that the unwanted features don’t affect the model training. This data is then split for training and testing the model. The model is trained using the training data and the trained model is test using various evaluation metrics for binary and multiclass classification of attack types.

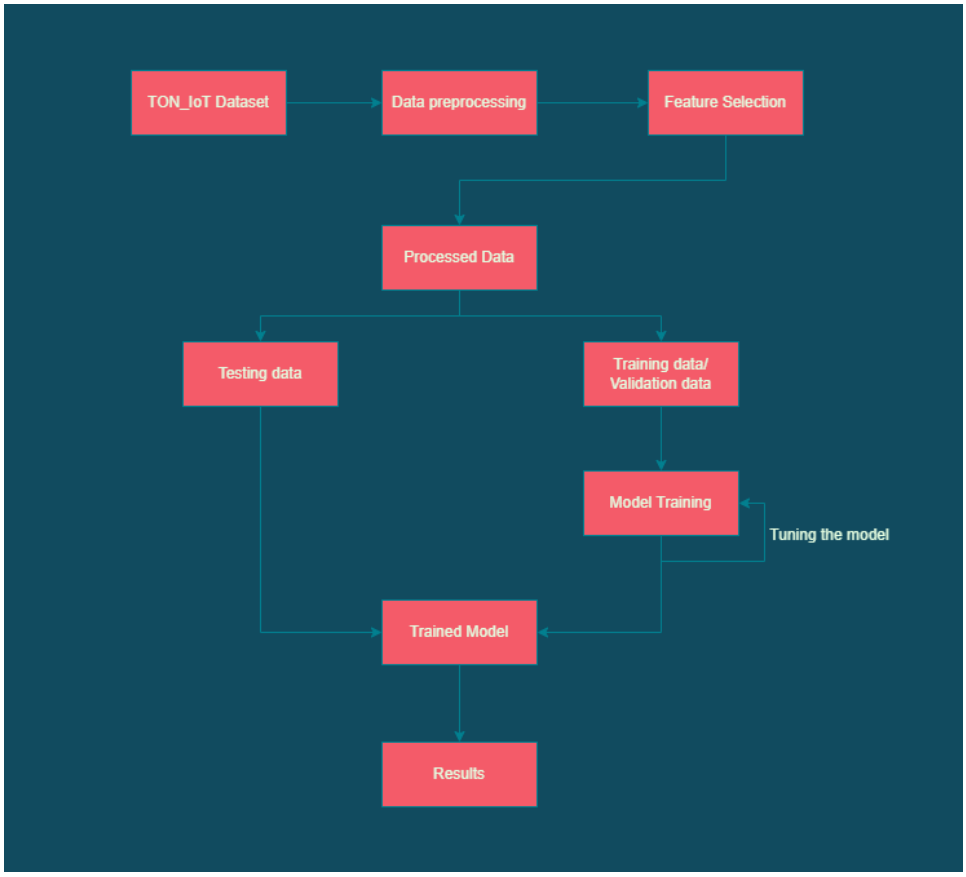


Figure 2 Overview of the proposed IDS

4.2 Data Preprocessing

The data preprocessing is the process of applying set of techniques on the dataset to prepare the data for providing it to the ML/DL model. The performance of the IDS does not only depend on the design of the ML/DL methods used but is also very dependent on the quality of the data. The quality of the data is affected by factors like noise, missing entities and inconsistent data influence the learning model hugely by providing false knowledge. Thus, data preprocessing plays vital stage when we require the final dataset to be consistent and relevant for the model to obtain essential knowledge (García *et al.*, 2016).

The TON_IoT dataset considered for this research has few inconsistent data which is required to be cleaned and prepared before feeding it to the DL model. The dataset contains 44 features along with the label and attack types. Among these features, the connection and the statistical activity features are the optimal features which contribute more for the learning process. Table 1 explains the features considered for this research from the TON_IoT network dataset. The following data preprocessing techniques are implemented based on the literature.

Table 1 Features extracted from TON_IoT dataset

Feature	Description
ts	Timestamp of connection
src_ip	Source IP address
src_port	Source port
dst_ip	Destination IP address
dst_port	Destination port
proto	Protocols of the transport layer
service	Dynamically detected protocols
duration	Time of packet connections i.e., subtracting "time of last seen packets" and "time of first seen packets"
src_bytes	Source bytes originated from payload bytes
dst_bytes	Destination bytes originated from payload bytes
conn_state	Connection states
missed_bytes	Number of missing bytes

src_pkts	Number of original packets estimated from source
src_ip_bytes	Number of original IP bytes which is the total length of IP header field of source
dst_pkts	Number of destination packets estimated from destination
dst_ip_bytes	Number of destination IP bytes which is the total length of IP header field of destination

4.2.1 Data Cleaning: The TON_IoT dataset contains numerous categorical features which is difficult for the IDS to learn and build a model based on it. These categorical features are converted into numerical features so that the DL model makes accurate predictions by calculating the correlation between features. There are various encoding techniques for converting the categorical features into numerical values of which one-hot encoding is the most common approach. The one-hot encoding is a type of encoding technique, where the original feature is extended to a multidimensional matrix, with each dimension of the matrix representing a certain state. The state of a particular dimension is given one while all other dimensions are zero. The major advantage of using one-hot encoding is that it takes care of imputation of missing data, where the missing values are considered as new class, which eliminates the process of treating the data by providing common value or mean value (Yu *et al.*, 2022).

Among the considered 16 features of TON_IoT dataset, only three features (conn_state, proto, service) has string data as values which needs to be encoded. The features were encoded into a matrix of dimension 13 for conn_state, 3 for proto and 10 for service. In

all the features, only the service feature class contains missing values denoted by "-" which was treated as separate class after encoding process. The Python library called pandas was used to implement the one-hot encoding using the *get_dummies()*function. The *get_dummies()* function encodes these three features and concatenates it to the dataset. Along with these features, the labels in the dataset were encoded into a separate dataframe for validation during the model learning process.

4.2.2 Class Imbalance: The class imbalance problems persist in datasets where the samples from one class is in higher number than the other classes. The TON_IoT dataset contains about 65% of normal network records while only 4% of a particular network attack record is present. To overcome this problem, the sampling techniques are widely used which artificially re-samples the dataset. The sampling can be carried out in two ways: Under-sampling and Over-sampling. In Under-sampling, the records of the majority class are removed randomly to balance the class distribution. This may lead to loss of valuable information. In Over-sampling, the records of the minority classes are replicated to balance the class distribution (Longadge *et al.*, 2013).

In this research, the works of (Lin *et al.*, 2017) are implemented to solve the class imbalance problem. The authors proposed the focal loss which address the class imbalance by down-weighting the well-classified classes. Since the majority class gets classified easily, the focal loss is introduced to provide more relative weight to the minority classes which will improve the accuracy of these classes. The Focal loss reshapes the loss function to down-weight the majority classes which results in the model focusing on hard negatives during training. The focal loss is defined as:

$$FL(p_t) = -\alpha_t(1 - p_t)^{\gamma} \log(p_t) \quad (1)$$

Where p is the model's estimated probability for the class, α is the weighting factor and γ is the tunable focusing parameter.

4.2.3 Feature Scaling: The TON_IoT dataset has features with various values, ranges and some features with values much greater than others. When these features are considered for training, the features with large range variable puts more weights leading to incorrect predictions. To overcome this, the feature scaling techniques are implemented to set all the features in the same scale which helps the ML/DL algorithm to consider each feature with equal importance. The two process of feature scaling are normalization and standardization. The data normalization commonly called as Max-Min Normalization is a technique to re-scale the feature values with a distribution between 0 and 1. The data standardization also called as Z-score normalization is a technique where the feature values are rescaled to ensure the mean and standard deviation to be 0 and 1, respectively (Ali *et al.*, 2014). In this research, the standardization technique is considered because of its robust nature to outliers. The equation by which data standardization is implemented is shown below.

$$data_{st} = \frac{data - mean(data)}{standard\_deviation(data)} \quad (2)$$

4.3 Feature Selection

The performance of ML/DL algorithms decreases gradually when the feature dimensions are high. This is because the higher dimension data includes lot of noisy and redundant data. The feature selection process is undergone to tackle these problems where the features which contributes more to the prediction variable are selected. By this process, all the information can be extracted from few features which contains maximum information about discriminating the classes. When a smaller number of features are considered, the ML/DL model have high explainability and less training time. The feature selection approaches are broadly classified into filter, wrapper, and embedded methods. In Filter methods, the features are ranked and the features with higher ranks are selected and provided to the model. Whereas in wrapper methods, the model is constructed with a search algorithm that selects the feature subset which provides high model performance. The Embedded methods include feature selection as part of the training process reducing the computation time (Chandrashekar and Sahin, 2014).

The Filter methods is considered widely because of its ranking techniques which is simple and produces good results for practical applications. The features with no influence on class labels are rejected by finding the inter correlation between the features. Pearson's Correlation (PC) and Mutual Information methods (Guyon and De, 2003) are commonly used filter methods which is based on statistical measures. The PC method is considered for this research. The PC is a method of finding linear relationship between two feature variables. The equation for calculating the correlation between features is provided below. The PC value lies between -1 and 1, where the value of -1 represents the features are negatively correlated and value of 1 represents the positive correlation. If the value is 0, there is no correlation between

them (Guyon and De, 2003).

$$R(i) = \frac{cov(x_i, Y)}{\sqrt{var(x_i) * var(Y)}} \quad (3)$$

where x_i is the i_{th} variable, Y is the output (class labels), cov() is the covariance and var() is the variance.

The 16 features shown in Table 1 were considered for this research which after data preprocessing were converted into 39 features. Before feature selection process, the source and destination IP addresses were discarded as the IP addresses can be easily spoofed by any attacker which makes it infeasible to use it as a feature for attack classification as stated by (Seo and Lee, 2016). The other 37 features were considered to calculate the correlation between them, and their relationship is shown in Figure 3 through correlation heatmap. From the correlation matrix, the features which are highly correlated (either positively or negatively) were dropped because they act as a duplicate feature where a change in one feature could influence another which makes the model unstable. The threshold of 0.9 was set to find features with high correlation to drop. Only the proto_udp feature was dropped because of its high negative correlation of -0.97 with the proto_tcp feature.

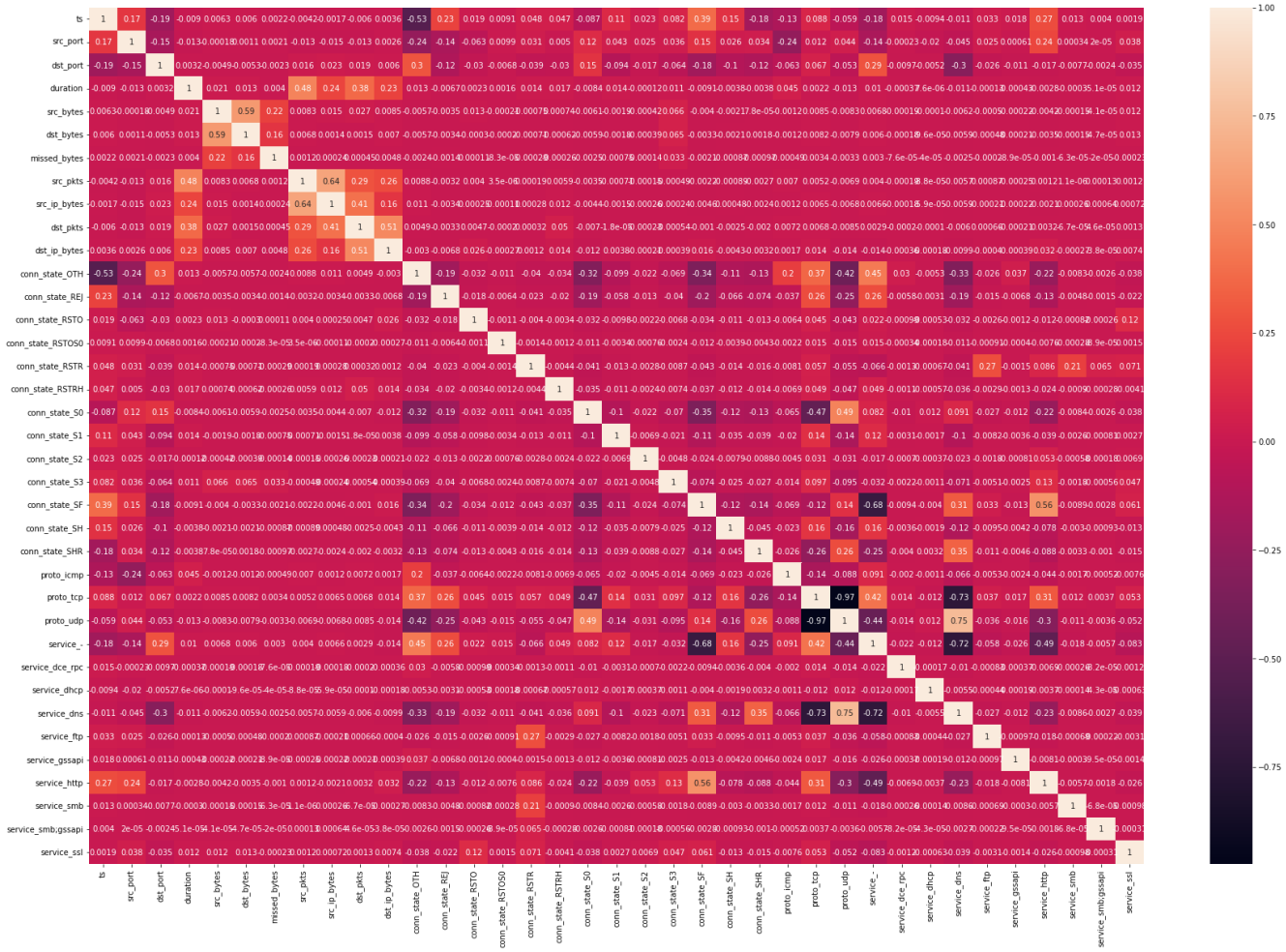


Figure 3 Correlation Heatmap of TON_IoT dataset features

4.4 Deep Learning algorithms

The Deep learning approaches are applied for this research because of its better performance and efficiency when compared to other ML methods. The two deep leaning algorithms implemented for detecting intrusions and attack classification using TON_IoT dataset are CNN-LSTM and Deep Residual 1D CNN.

4.4.1 Convolutional Neural Networks (CNN) – LSTM: The CNN-LSTM model consists of two components; one is CNN which develops the features of the input data, and another is LSTM which exploits the generated features. The CNN is a feed forward network mostly used for image classification, while the LSTM is a recurrent network which is mostly used for time series data. The CNN-LSTM architecture consists of multiple layers as shown in Figure 4. The convolutional layer, pooling layer, and fully connected layer are the three basic layers that make up the CNN algorithm.

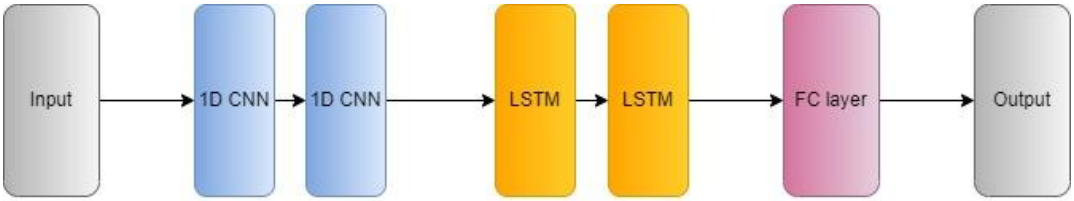


Figure 4 The proposed hybrid CNN-LSTM model

4.4.1.1 Convolutional Layer: The convolutional layer functions with the use of kernel which is a spatial dimensionality that moves across the entire input volume while calculating scalar product of weights and the region connected to the input volume. The three hyperparameters used for optimizing the output of the convolutional layers are depth, stride and padding. The depth determines the number of neurons of the network which is essential for pattern recognition. The stride determines the number of nodes the filter window moves around the input volume. If the stride value is set low, heavily overlapped outputs are generated which produces large activations. Padding is a process of adding value across the borders of the input which helps in retaining the loss of information during convolution process (O'Shea and Nash, 2015). The convolutional layer is processed using the formula provided below

$$x_i = f(w_i \otimes x_{i-1} + b_i) \quad (4)$$

where x is the sample of training input data, w_i is the weighted matrix, x_{i-1}, is the sample of training input data, $\otimes$ is the convolution operation, f is the activation function, and b_i is the basis of neural network.

4.4.1.2 Pooling Layer: The pooling layer is present to reduce the dimensionality of the considered filter window of the input which reduces the computational complexity of the model. The commonly used pooling layer is the max pooling layer which reduces the parameter in the feature map by selecting the maximum value in each region (O'Shea and Nash, 2015).

4.4.1.3 Fully connected Layer: The fully connected layer serves as a representation of the convolutional neural network's final layer. Every node in the layer that is fully connected is directly connected to every node in layers (L 1) and (L + 1). In contrast to the conventional ANN, there is no link between nodes in the same layer. As a result, training and testing for this layer are time-consuming.

4.4.1.4 Long Short-Term Memory (LSTM): The LSTM networks were first introduced by (Hochreiter and Schmidhuber, 1997) that overcomes the vanishing gradient problem faced by RNN. LSTMs differ from more conventional feedforward neural networks in that they feature feedback connections. With the help of this property, LSTMs may process whole data sequences without considering each data point separately. Instead, they can process new data points by using the information from earlier data in the sequence. A series of "gates" used by LSTMs regulate how data in a sequence enters, is stored in, and leaves the network. A typical

LSTM has three gates: an output gate, an input gate, and a forget gate. Figure 5 shows the structure of the LSTM model. Firstly, the forget gate decides which information to save or discard through sigmoid activation function where the values closer to 0 are regarded as irrelevant and the values close to 1 are relevant. Secondly, the new memory network and input gate determines the new information that needs to be added to the long-term memory for the previous hidden state and new input data. The output gate decides the new hidden state given the new cell state, previous hidden state, and new input data (Dolphin, 2020).

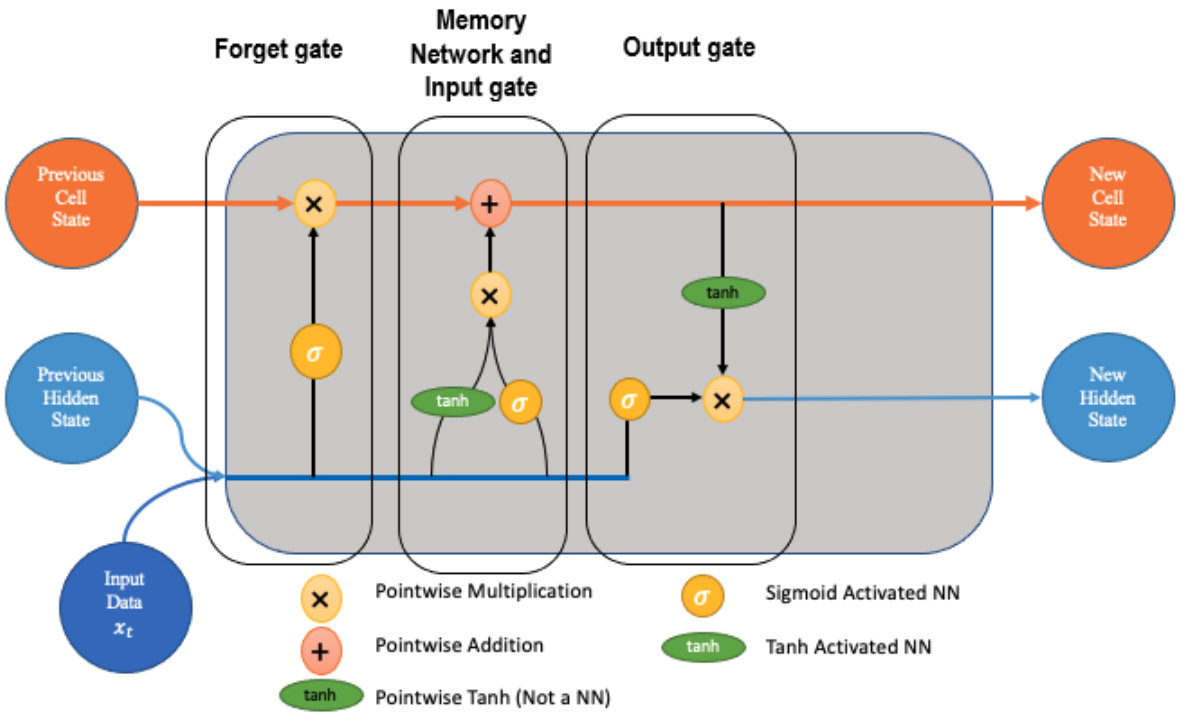


Figure 5 Architecture of LSTM

In this research, the hybrid CNN-LSTM model shown in Figure 4 was implemented to detect intrusions and to classify the attacks. The two 1D convolutional layers were created using the Conv1D() function from keras with the kernel size as 5, strides as 1 and with no padding. The first layer has the filter values of 64 whereas the second one with 32. The two LSTM hidden layers are present which returns a sequence of vectors of dimensions 32 and 16 respectively. The output of the LSTM layer is flattened and provided to the two dense layer of output dimensions 128 and 64 respectively. The element-wise activation function for the dense layers is applied using a nonlinear activation function called a rectified linear (ReLU). For negative values, the ReLU function returns 0, and for positive values, it returns any value x. The range of the ReLU function is 0 to infinity. The Softmax function is utilised as the activation for the last layer of a classification network since the outcome could be interpreted as a probability distribution. It transforms a value vector into a probability distribution. The output vector's members are in the range (0, 1) and add up to 1. Figure 6 shows the model structure of the proposed CNN-LSTM model.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 36, 64)            384

 conv1d_1 (Conv1D)           (None, 36, 32)            10272

 lstm (LSTM)                 (None, 36, 32)            8320

 lstm_1 (LSTM)               (None, 36, 16)            3136

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 128)               73856

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 104,874
Trainable params: 104,874
Non-trainable params: 0
_________________________________________________________________
```

Figure 6 Structure of CNN-LSTM model

4.4.2 Deep Residual CNN (DRCNN): The residual network has had considerable success in the field of image processing since it was first presented by (He *et al.*, 2016). The residual network often has three key characteristics. First, the identity skip-connections are established, allowing data to flow directly from other layers to the following layers. Second, by using skip connections, the depth of the network structure can be significantly increased. Finally, the performance of the tests is essentially unaffected by eliminating single layers from residual networks (Zhang, Li and Ding, 2019). Figure 7 depicts a residual learning block that is described as

$$z = F(x, \{w_i\}) + x \tag{5}$$

Where x and z denote the input and output vectors of the layer, respectively. F represents the residual function. The residual function shown in the figure is defined as $F = w_2\varphi(w_1^T.x)$, where φ is the nonlinear activation function.

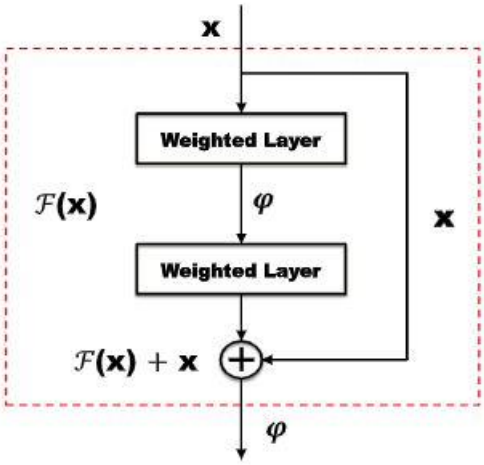


Figure 7 One block of residual network (Zhang, Li and Ding, 2019)

In this research, a five stacked 1D CNN was implemented for the proposed deep residual CNN model. Figure 8 shows the proposed DRCNN model for attack classification of network intrusions. Each building block of the residual network consist of two 1D CNN with filter window of 64, kernel size of 5, strides as 1 and with no padding. The ReLU function is applied to each of the convolution as the activation function. The Maximum pooling layers are applied at end of each block of the residual network which reduces the dimensions to accelerate the training process while retaining the important information. The five stacked convolutional layers are then flattened and provided to fully connected layer which classifies the attack using the sigmoid activation function. Figure 8 shows the structure of the proposed DRCNN model, where the dimensions of the output at each layer are present along with the number of parameters.

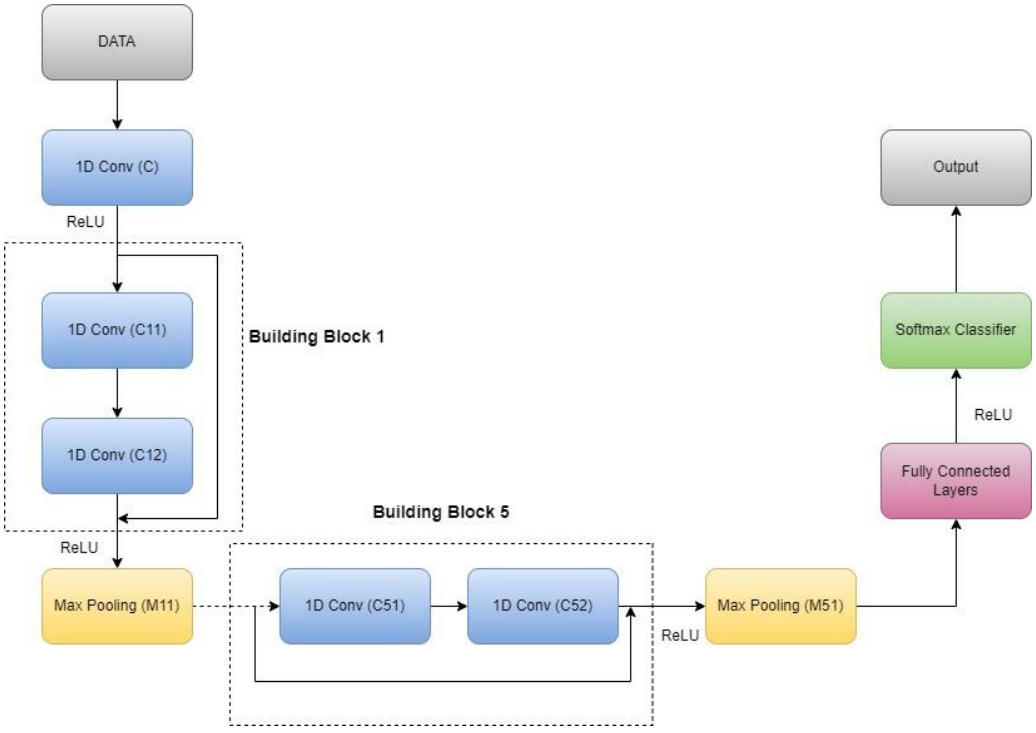


Figure 8 The architecture of the proposed DRCNN model

input_1 (InputLayer)	[(None, 36, 1)]	0	[]
conv1d_2 (Conv1D)	(None, 32, 64)	384	['input_1[0][0]']
conv1d_3 (Conv1D)	(None, 32, 64)	20544	['conv1d_2[0][0]']
activation (Activation)	(None, 32, 64)	0	['conv1d_3[0][0]']
conv1d_4 (Conv1D)	(None, 32, 64)	20544	['activation[0][0]']
add (Add)	(None, 32, 64)	0	['conv1d_4[0][0]', 'conv1d_2[0][0]']
activation_3 (Activation)	(None, 32, 64)	0	['add[0][0]']
max_pooling1d_1 (MaxPooling1D)	(None, 15, 64)	0	['activation_3[0][0]']
conv1d_7 (Conv1D)	(None, 15, 64)	20544	['max_pooling1d_1[0][0]']
activation_4 (Activation)	(None, 15, 64)	0	['conv1d_7[0][0]']
conv1d_8 (Conv1D)	(None, 15, 64)	20544	['activation_4[0][0]']
add_2 (Add)	(None, 15, 64)	0	['conv1d_8[0][0]', 'max_pooling1d_1[0][0]']
activation_5 (Activation)	(None, 15, 64)	0	['add_2[0][0]']
max_pooling1d_2 (MaxPooling1D)	(None, 7, 64)	0	['activation_5[0][0]']
conv1d_9 (Conv1D)	(None, 7, 64)	20544	['max_pooling1d_2[0][0]']
activation_6 (Activation)	(None, 7, 64)	0	['conv1d_9[0][0]']
conv1d_10 (Conv1D)	(None, 7, 64)	20544	['activation_6[0][0]']
add_3 (Add)	(None, 7, 64)	0	['conv1d_10[0][0]', 'max_pooling1d_2[0][0]']
activation_7 (Activation)	(None, 7, 64)	0	['add_3[0][0]']
max_pooling1d_3 (MaxPooling1D)	(None, 3, 64)	0	['activation_7[0][0]']
conv1d_11 (Conv1D)	(None, 3, 64)	20544	['max_pooling1d_3[0][0]']
activation_8 (Activation)	(None, 3, 64)	0	['conv1d_11[0][0]']
conv1d_12 (Conv1D)	(None, 3, 64)	20544	['activation_8[0][0]']
add_4 (Add)	(None, 3, 64)	0	['conv1d_12[0][0]', 'max_pooling1d_3[0][0]']
activation_9 (Activation)	(None, 3, 64)	0	['add_4[0][0]']
max_pooling1d_4 (MaxPooling1D)	(None, 1, 64)	0	['activation_9[0][0]']
flatten_1 (Flatten)	(None, 64)	0	['max_pooling1d_4[0][0]']
dense_3 (Dense)	(None, 32)	2080	['flatten_1[0][0]']
activation_10 (Activation)	(None, 32)	0	['dense_3[0][0]']
dense_4 (Dense)	(None, 32)	1056	['activation_10[0][0]']
dense_5 (Dense)	(None, 10)	330	['dense_4[0][0]']
activation_11 (Activation)	(None, 10)	0	['dense_5[0][0]']

Total params: 168,202
Trainable params: 168,202
Non-trainable params: 0

Figure 9 The structure of DRCNN model

4.5 Evaluation Metrics

The model's performance and accuracy must be assessed after it has been properly hyper tuned and trained. In this study, the roc curve and a classification report were utilised as parameters to evaluate the model's performance.

4.5.1 Classification Report

Slearn.metrics' classification report calculates accuracy, precision, recall, and f1-score along with the confusion matrix.

a. Accuracy: It is described as the proportion of accurate predictions among all predictions. Its mathematical formula is

$$Accuracy = Correct\ prediction\ /\ Total\ number\ of\ datasets.$$

b. Precision: According to the prediction outcomes, it shows how many forecasts made for that class were accurate. The following statement provides it as

$$Precision = True\ Postive\ /\ (True\ Positive + False\ Positive)$$

c. Recall: The following formula gives the amount by which the classifier detects a particular class:

$$Recall = True\ Postive\ /\ (True\ Positive + False\ Negative)$$

d. F-1 Score: It is the harmonic mean of recall and precision.

$$F1\ Score = 2 * (Precision * RecallPrecison + Recall)$$

e. Confusion Matrix: An all-in-one tool for tracking model performance on the unbalanced dataset is the confusion matrix. For each class, it offers instructions on how to calculate true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). It is possible to determine accuracy, recall, precision, recall, and F1-score using the count. Figure 10 shows a sample confusion matrix with markings for TP, TN, FP, and FN for

a class.

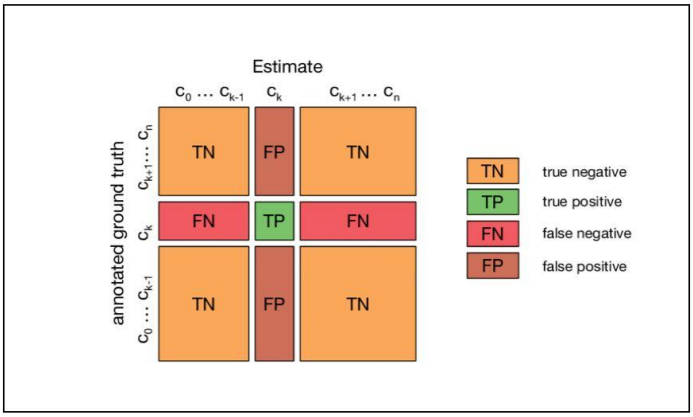


Figure 10 Illustration of confusion matrix (Krüger, 2016)

4.5.2 ROC Curve

Receiver operating characteristic, or ROC, is well known for visualising the trade-offs that might be made while training a model. It is a plot of True Positive Rate vs False Positive Rate at various thresholds. TPR is also known as sensitivity, and FPR is known as 1 – (true negative rate or specificity). The area under the curve (ROC AUC), which is frequently used to calculate the ROC Curve, is illustrated as an example in Figure 11.

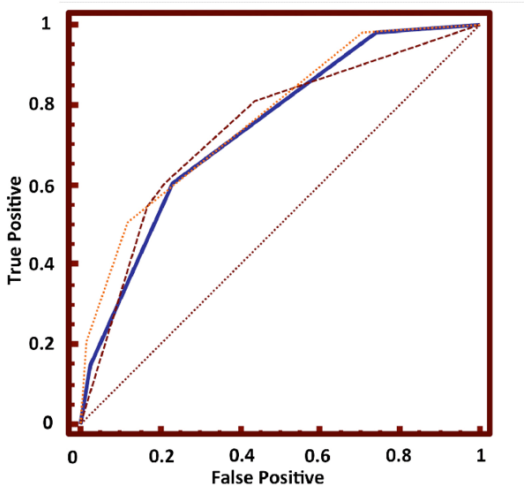


Figure 11 Interpretation of ROC curve (Ferraz Cavalcanti et al.*, 2015)*

4.6 Implementation

The two deep learning approaches were implemented to detect intrusions and to classify the attack types in the TON_IoT dataset. The IDS was built on Google Colaboratory which is a data analysis and machine learning tool. The TON_IoT network dataset was imported, and the required software and libraries were initialized. The experimental setup of this research with the hardware and software implementations are provided in the Appendix II. The pandas' read_csv() function was used to read the network dataset which is in csv format. The imported data frame was shuffled using NumPy package to avoid any pattern in the dataset before training. The labels in the dataset were encoded and stored separately.

In the Data preprocessing stage, the required features are selected while the other were dropped. The feature with categorical values were converted into numerical values through one-hot encoding. Once the features are encoded, the feature values are scaled through standardization process. A standardize() function was created for this purpose. The corr() function from pandas' package were utilized to find correlation between the features and the features with high correlation were dropped. A focal loss function was created to modify the loss function of the training process which tackles the class imbalance problems. Finally, the standardized features were split for training and testing purposes. For training the model, 70% of the dataset was provided of which 12.5% were provided for validation. 30% of the data were considered for testing the model.

The CNN-LSTM deep learning algorithm was first implemented with two convolutional layer and LSTM hidden layers each. Secondly, the DRCNN model with five 1D CNN were implemented for attack classification. A model_val() function was created which

compiles and fits both the deep learning models. For training the networks, Adam optimization method is used with the learning rate, beta-1, and beta-2 of 0.001, 0.9, and 0.999, respectively. The Adam optimizer and the focal loss defined above were passed for the model compilation process. After multiple execution process, the epochs and batch size of the model training process were set to 70 and 512 respectively. Learning rate is decayed exponentially with the decay factor of 0.5 every 100000 iterations. The ReduceLROnPlateau() function was utilised for reducing the learning rate and the EarlyStopping() function was used to stop the training process when there is no improvement in the results. Finally, the models were evaluated using the evaluation metrics discussed above and their results were recorded.

5 RESULTS AND DISCUSSION

5.1 Experimental Results

This section discusses the results attained from different simulation experiments to provide understanding of the proposed DRCNN and CNN-LSTM models.

5.1.1 Experiment 1: The performance of the CNN-LSTM model was evaluated by calculating the accuracy, precision, recall and F1-score. Figure 12 shows the metrics of the model in classifying the attack types. The overall performance of the CNN-LSTM model shows 99.64% in precision, recall and f1-score. The accuracy and loss of the model when tested with the testing dataset is 99.64% and 0.023%. The confusion matrix was plotted with True class and the predicted class. Figure 13 shows the confusion matrix with the number of attack types and its percentage after attack classification. The accuracy in classifying the attacks over the number of epochs during the training phase is provided in Figure 14. The cross-entropy loss over the number of epochs for training the model is also presented in Figure 14.

```
              precision    recall  f1-score   support

    backdoor     0.9998    1.0000    0.9999      5992
        ddos     0.9997    0.9988    0.9992      5990
         dos     0.9998    0.9990    0.9994      6012
   injection     0.9993    0.9995    0.9994      5931
        mitm     1.0000    1.0000    1.0000       302
      normal     0.9975    0.9971    0.9973     90098
    password     0.9987    0.9892    0.9939      6036
  ransomware     0.9882    0.9897    0.9889      5905
    scanning     0.9995    1.0000    0.9998      6018
         xss     0.9712    0.9844    0.9778      6029

    accuracy                         0.9964    138313
   macro avg     0.9954    0.9958    0.9956    138313
weighted avg     0.9964    0.9964    0.9964    138313

4323/4323 [==============================] - 44s 10ms/step - loss: 2.3045e-04 - accuracy: 0.9964
Test: accuracy = 0.996428  ;  loss = 0.000230
```

Figure 12 Performance metrics of the CNN-LSTM model

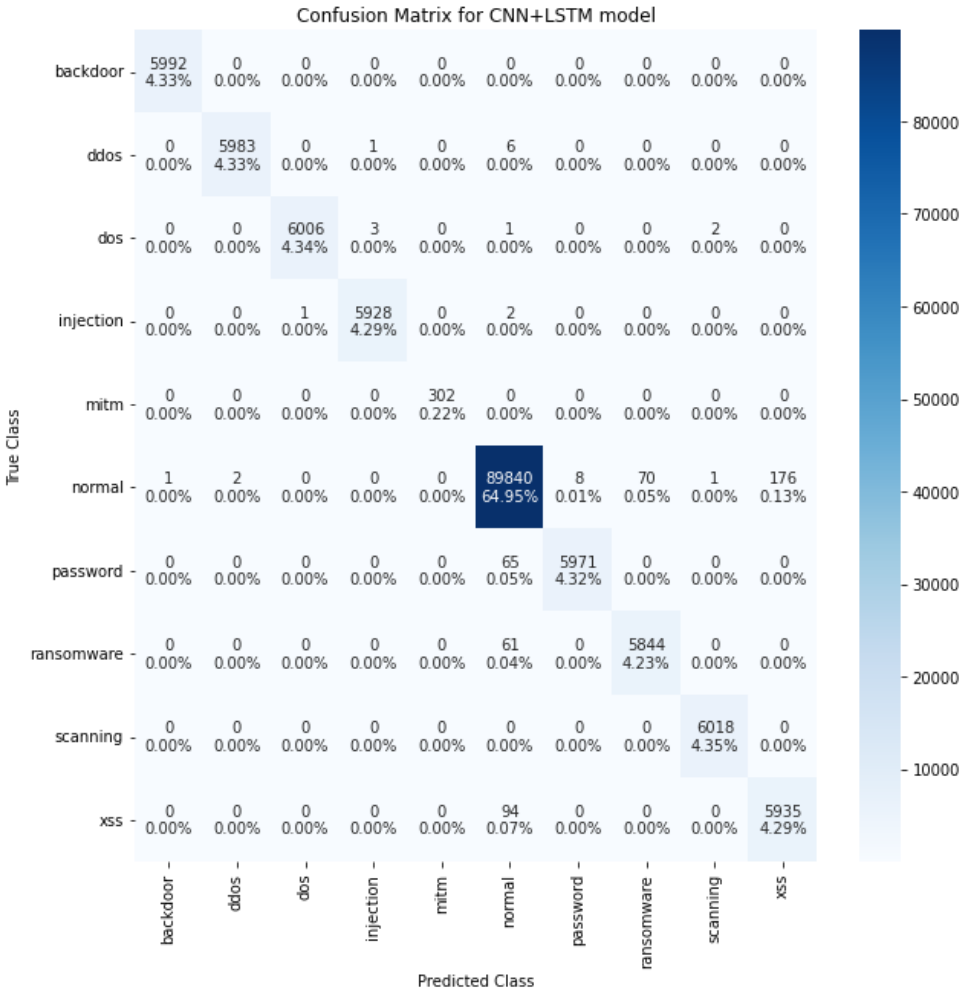


Figure 13 Confusion matrix for CNN-LSTM model

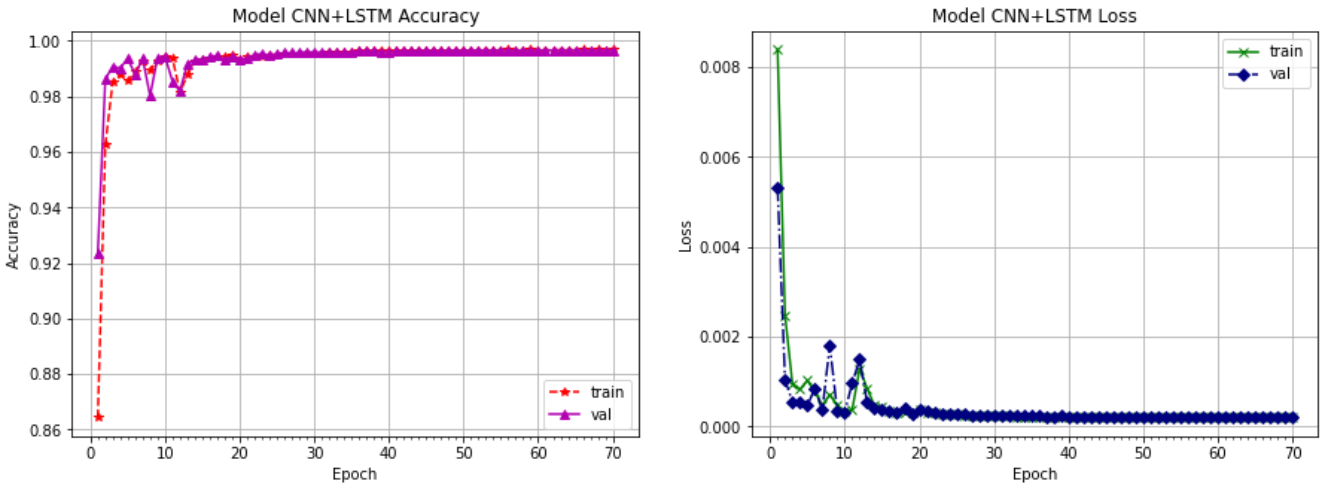


Figure 14 Accuracy and loss during the CNN-LSTM training process

5.1.2 Experiment 2: In this experiment, the proposed DRCNN model was evaluated using the testing data. The weighted average of the precision, recall and f1-score evaluated after testing the model is 99.78%, 99.77% and 99.77% respectively. The evaluation results of the model are presented in Figure 15. The accuracy and loss calculated during the training process is shown in Figure 17. The accuracy showed some fluctuations during the early epochs between 94% and 98% then it attained saturation level around 99.7%. The confusion matrix for the DRCNN model is provided in Figure 16. The overall accuracy and loss in detecting and classifying the attack types of the testing data is 99.77% and 0.013%. This shows that the DRCNN model performed better in classifying the attack than the CNN-LSTM model.

	precision	recall	f1-score	support
backdoor	1.0000	1.0000	1.0000	5992
ddos	0.9998	0.9997	0.9997	5990
dos	1.0000	0.9997	0.9998	6012
injection	0.9997	0.9993	0.9995	5931
mitm	1.0000	1.0000	1.0000	302
normal	0.9985	0.9981	0.9983	90098
password	0.9993	0.9899	0.9946	6036
ransomware	0.9885	0.9900	0.9893	5905
scanning	1.0000	0.9998	0.9999	6018
xss	0.9833	0.9983	0.9908	6029
accuracy			0.9977	138313
macro avg	0.9969	0.9975	0.9972	138313
weighted avg	0.9978	0.9977	0.9977	138313

```
4323/4323 [==============================] - 30s 7ms/step - loss: 1.3823e-04 - accuracy: 0.9977
Test: accuracy = 0.997744  ;  loss = 0.000138
```

Figure 15 Evaluation metrics of DRCNN model

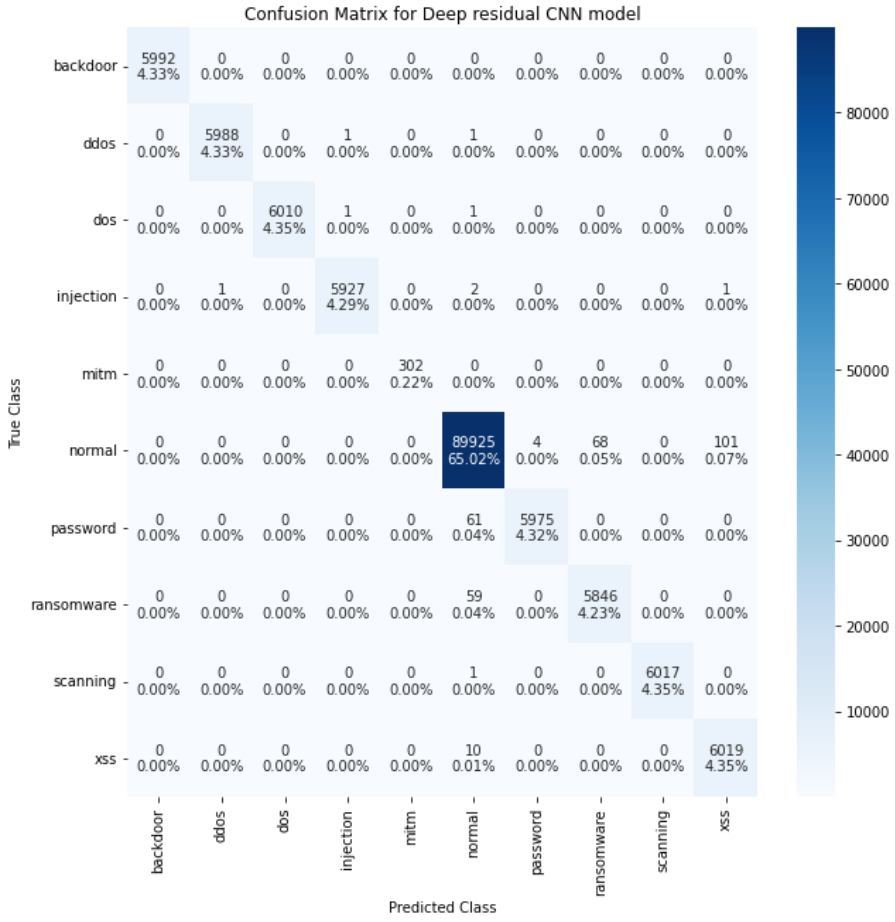


Figure 16 Confusion matrix for DRCNN model

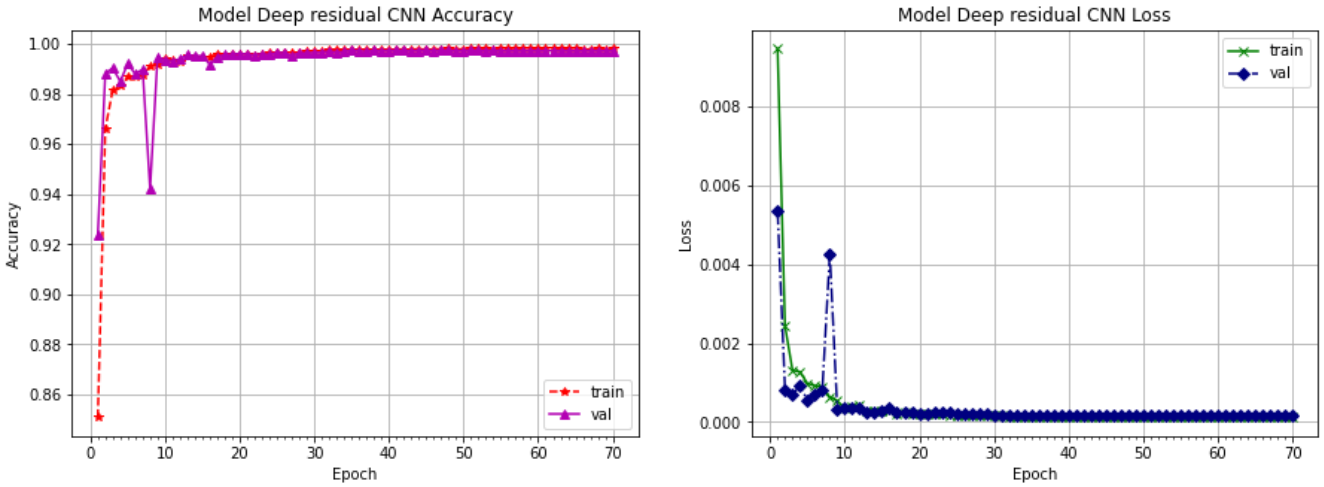


Figure 17 Accuracy and loss during the DRCNN training process

Figure 18 and 19 shows the receiver operating characteristic (ROC) curves for experimental results of both the models in classifying the attack types. The ROC is used to measure the validation of the proposed system to detect intrusions and classify the attacks from IoT devices. The graphical representation (y-axis) is the recall metric for detecting nine attacks and normal network traffic in the IoT network; x-axis represents the specificity metric for detecting all the attack types.

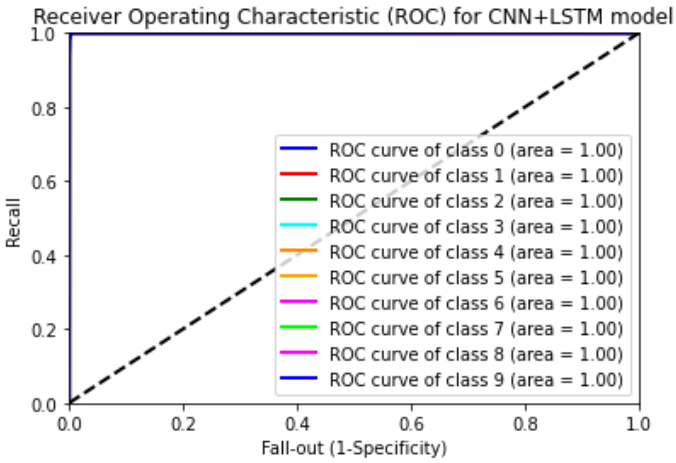


Figure 18 ROC curve for CNN-LSTM model

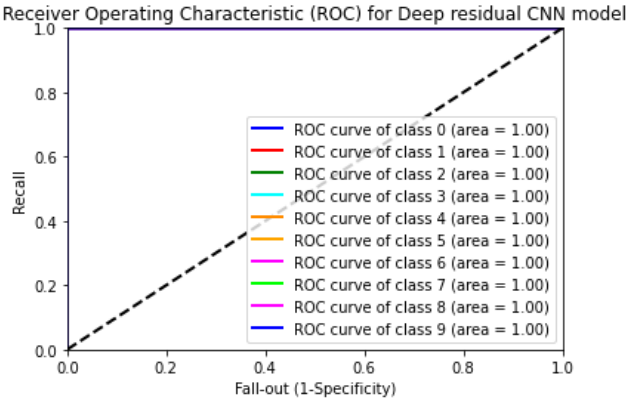


Figure 19 ROC for DRCNN model

5.2 Discussion

Massive improvements in telecommunications networks and the emergence of the Internet of Things have resulted from incredible changes in the ordinary use of electronic services and applications. However, IoT systems are vulnerable to a number of security threats, including DDoS, DoS, Injection attacks and MITM attacks. Such attacks have the potential to seriously harm an IoT network's applications for smart environments and IoT services. As a result, protecting IoT systems has grown to be a top priority (Elrawy, Awad and Hamed, 2018). An IoT platform can therefore be effectively protected by utilising artificial intelligence models to detect attacks by extracting numerous unknown patterns created by actors. Two deep learning techniques were implemented in this study to detect and classify attacks in the IoT network. The following observations can be noted by analysing the experimental results of both the deep learning models.

- Both the deep learning approaches performed effectively in detecting intrusions with the accuracy of more than 99.5%.
- Introduction of focal loss for class imbalance problems supported in detecting classes like mitm attack which constitutes to less than 0.3% of the overall data, with 100% precision.
- Both the model showed few False positive for normal traffic as ransomware and xss attacks. There were very few True negative for password and ransomware attack which were detected as normal traffic.
- The attack type which showed least precision, recall and f1-score for both the models was xss attack. Although, its metrics did not fall below 97%.
- Both the model's ROC curve demonstrates that 100% accuracy is attained in

identifying all the attacks.

Overall, the DRCNN model performed more effectively in classifying the attack types compared to the CNN-LSTM. Although the proposed system was able to perform with high detection rates, the system needs to be optimized to reduce the execution time where the DRCNN model consumed resources and time than the CNN-LSTM model.

The proposed system was compared with the IDS discussed in the literature. The majority of the IDS discussed in the literature focuses on improving the accuracy of the system while employing varies ML/DL methods. By doing so, they oversee the computational cost and high false alarm rates. Most studies did not follow basic data preprocessing procedures, which are crucial for cleaning the data so that the model can find significant patterns and generate accurate predictions. Table 2 presents the comparison of the proposed DRCNN with the state-of-the-art methods. The proposed model achieves the most accurate threat detection results in comparison to other IDS. We could observe that the accuracy, precision, recall, and F-score of the model, 99.77%, 99.78%, 98.77%, and 99.77% respectively, which is far better than the other methods. We can readily conclude that the DRCNN detection model is suitable for managing massive, highly dimensional data generated by the networks of IoT.

Table 2 Comparison of proposed system with the IDS in the literature

Model	Accuracy	Precision	Recall	F1-Score
(Khan *et al.*, 2022)	99.38	99.39	98.99	99.37
(Gad, Nashat and Barkat, 2021)	97.8	97.8	97.8	97.8
(Latif *et al.*, 2021)	99.05	99.13	99.08	99.17
(Lo *et al.*, 2022)	87.78	87.78	87.78	87.78
(Abdel-Basset *et al.*, 2022)	97.04	-	-	96.56
DRCNN model	99.77	99.78	99.77	99.77

6 CONCLUSION

This research presents an IDS for IoT based on TON_IoT using deep learning approaches. The two deep learning approaches implemented in this work is CNN-LSTM and DRCNN algorithm. The TON_IoT dataset was considered because of its heterogenous data collected from the IIoT/IoT network. The problems of the dataset having class imbalance issues and categorical values were addressed by implementing focal loss function to deal class imbalance and one-hot encoding for converting categorical values to numeric values for the model to easily process. To achieve high accuracy and efficiency, the features selection process was carried out using the Pearson's Correlation technique, where the highly correlated features were dropped. The CNN-LSTM model was trained which was able to achieve training accuracy and loss of 99.68% and 0.02%. Whereas the DRCNN model achieved 99.81% and 0.015% as training accuracy and loss. The models were evaluated using the metrics like precision, recall and f1-score, and the results for the CNN-LSTM model was 99.64% each and for the DRCNN model it was 99.78%, 99.77% and 99.77% respectively. After evaluating both the DL model, the DRCNN model has the best performance in multi-class classification of attacks compared to other state-of-the-art methods. In future works, the DRCNN model can be optimized to improve the resource and time consumption during the training process.

7 REFERENCE

Abdel-Basset, M. *et al.* (2022) 'Federated Intrusion Detection in Blockchain-Based Smart Transportation Systems', *IEEE Transactions on Intelligent Transportation Systems*, 23(3), pp. 2523–2537. Available at: https://doi.org/10.1109/TITS.2021.3119968.

Ahmad, Z. *et al.* (2020) 'Network intrusion detection system: A systematic study of machine learning and deep learning approaches'.

Albin, E. and Rowe, N.C. (2012) 'A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems', in *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, pp. 122–127. Available at: https://doi.org/10.1109/WAINA.2012.29.

Ali, Peshawa Jamal Muhammad *et al.* (2014) 'Data normalization and standardization: a technical report', *Mach Learn Tech Rep*, 1(1), pp. 1–6.

Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A. and Anwar, A. (2020) *TON_IoT Dataset, IEEE Access*. Available at: https://cloudstor.aarnet.edu.au/plus/s/ds5zW91vdgjEj9i?path=%2F (Accessed: 23 August 2022).

Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A. and Adna N Anwar (2020) 'TON-IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems', *IEEE Access*, 8, pp. 165130–165150. Available at: https://doi.org/10.1109/ACCESS.2020.3022862.

Amini, P., Araghizadeh, M.A. and Azmi, R. (2015) 'A survey on Botnet: Classification, detection and defense', in *2015 International Electronics Symposium (IES)*, pp. 233–238. Available at: https://doi.org/10.1109/ELECSYM.2015.7380847.

ben Amor, N., Benferhat, S. and Elouedi, Z. (2004) 'Naive Bayes vs Decision Trees in Intrusion Detection Systems', *ACM Symposium on Applied Computing* [Preprint].

Andresini, G. *et al.* (2020) 'Multi-Channel Deep Feature Learning for Intrusion Detection', *IEEE Access*, 8, pp. 53346–53359. Available at: https://doi.org/10.1109/ACCESS.2020.2980937.

'Aposemat IoT-23' (2022). Available at: https://www.stratosphereips.org/datasets-iot23 (Accessed: 9 August 2022).

Ashton, K. (2009) *That 'Internet of Things' Thing*, *That 'Internet of Things' Thing-RFID Journal*.

Bhati, B.S. and Rai, C.S. (2020) 'Analysis of Support Vector Machine-based Intrusion Detection Techniques', *Arabian Journal for Science and Engineering*, 45(4), pp. 2371–2383. Available at: https://doi.org/10.1007/s13369-019-03970-z.

Booij, T.M. *et al.* (2022) 'ToN_IoT: The Role of Heterogeneity and the Need for Standardization of Features and Attack Types in IoT Network Intrusion Data Sets', *IEEE Internet of Things Journal*, 9(1), pp. 485–496. Available at: https://doi.org/10.1109/JIOT.2021.3085194.

Boser, B.E., Guyon, I.M. and Vapnik, V.N. (1992) 'A Training Algorithm for Optimal Margin Classifiers', in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. New York, NY, USA: Association for Computing Machinery (COLT '92), pp. 144–152. Available at: https://doi.org/10.1145/130385.130401.

Chandrashekar, G. and Sahin, F. (2014) 'A survey on feature selection methods', *Computers & Electrical*

Engineering, 40(1), pp. 16–28. Available at: https://doi.org/https://doi.org/10.1016/j.compeleceng.2013.11.024.

Chittur, A. (2001) 'Model generation for an intrusion detection system using genetic algorithms', *High School Honors Thesis, Ossining High School. In cooperation with Columbia Univ* [Preprint].

da Costa, K.A.P. *et al.* (2019) 'Internet of Things: A survey on machine learning-based intrusion detection approaches', *Computer Networks*, 151, pp. 147–157. Available at: https://doi.org/https://doi.org/10.1016/j.comnet.2019.01.023.

Dietterich, T.G. (2000) 'Ensemble Methods in Machine Learning', in *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–15.

Diro, A.A. and Chilamkurti, N. (2018) 'Distributed attack detection scheme using deep learning approach for Internet of Things', *Future Generation Computer Systems*, 82, pp. 761–768. Available at: https://doi.org/https://doi.org/10.1016/j.future.2017.08.043.

Dolphin, R. (2020) 'LSTM Networks | A Detailed Explanation'. Available at: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 (Accessed: 8 September 2022).

Dong, S., Wang, P. and Abbas, K. (2021) 'A survey on deep learning and its applications', *Computer Science Review*, 40, p. 100379. Available at: https://doi.org/https://doi.org/10.1016/j.cosrev.2021.100379.

Douligeris, C. and Mitrokotsa, A. (2003) 'DDoS attacks and defense mechanisms: a classification', in *Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology (IEEE Cat. No.03EX795)*, pp. 190–193. Available at: https://doi.org/10.1109/ISSPIT.2003.1341092.

Elrawy, M.F., Awad, A.I. and Hamed, H.F.A. (2018) 'Intrusion detection systems for IoT-based smart environments: a survey', *Journal of Cloud Computing*, 7(1), p. 21. Available at: https://doi.org/10.1186/s13677-018-0123-6.

Farahnakian, F. and Heikkonen, J. (2018) 'A deep auto-encoder based approach for intrusion detection system', in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pp. 178–183. Available at: https://doi.org/10.23919/ICACT.2018.8323688.

Feily, M., Shahrestani, A. and Ramadass, S. (2009) 'A Survey of Botnet and Botnet Detection', in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pp. 268–273. Available at: https://doi.org/10.1109/SECURWARE.2009.48.

Ferraz Cavalcanti, P. *et al.* (2015) 'Stratification of complexity in congenital heart surgery: Comparative study of the Risk Adjustment for Congenital Heart Surgery (RACHS-1) method, Aristotle basic score and Society of Thoracic Surgeons-European Association for Cardio- Thoracic Surgery (STS-EACTS) mortality score', *Revista Brasileira de Cirurgia Cardiovascular*, 30, pp. 148–158. Available at: https://doi.org/10.5935/1678-9741.20150001.

Finke, T. *et al.* (2021) 'Autoencoders for unsupervised anomaly detection in high energy physics', *Journal of High Energy Physics*, 2021(6), p. 161. Available at: https://doi.org/10.1007/JHEP06(2021)161.

Fischetti, M. (2016) 'Fast training of Support Vector Machines with Gaussian kernel', *Discrete Optimization*, 22, pp. 183–194. Available at: https://doi.org/https://doi.org/10.1016/j.disopt.2015.03.002.

Gad, A.R., Nashat, A.A. and Barkat, T.M. (2021) 'Intrusion Detection System Using Machine Learning for

Vehicular Ad Hoc Networks Based on ToN-IoT Dataset', *IEEE Access*, 9, pp. 142206–142217. Available at: https://doi.org/10.1109/ACCESS.2021.3120626.

Gao, N. *et al.* (2014) 'An Intrusion Detection Model Based on Deep Belief Networks', in *2014 Second International Conference on Advanced Cloud and Big Data*, pp. 247–252. Available at: https://doi.org/10.1109/CBD.2014.41.

García, S. *et al.* (2016) 'Big data preprocessing: methods and prospects', *Big Data Analytics*, 1(1). Available at: https://doi.org/10.1186/s41044-016-0014-0.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) 'Deep learning'.

Guyon, I. and De, A.M. (2003) *An Introduction to Variable and Feature Selection André Elisseeff*, *Journal of Machine Learning Research*.

Gyanchandani, M., Rana, J. and Yadav, R. (2012) 'Taxonomy of Anomaly Based Intrusion Detection System: A Review', *International Journal of Scientific and Research Publications*, 2(12).

Haghighat, A.T. *et al.* (2007) 'Intrusion Detection via Fuzzy-Genetic Algorithm Combination with Evolutionary Algorithms', in *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*, pp. 587–591. Available at: https://doi.org/10.1109/ICIS.2007.124.

He, K. *et al.* (2016) *Deep Residual Learning for Image Recognition*. Available at: http://image-net.org/challenges/LSVRC/2015/.

Hoang, X.D. and Nguyen, Q.C. (2018) 'Botnet Detection Based On Machine Learning Techniques Using DNS Query Data', *Future Internet*, 10(5). Available at: https://doi.org/10.3390/fi10050043.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation*, 9(8), pp. 1735–1780. Available at: https://doi.org/10.1162/neco.1997.9.8.1735.

Horng, S.-J. *et al.* (2011) 'A novel intrusion detection system based on hierarchical clustering and support vector machines', *Expert Systems with Applications*, 38(1), pp. 306–313. Available at: https://doi.org/https://doi.org/10.1016/j.eswa.2010.06.066.

Jia, Y., Wang, M. and Wang, Y. (2019) 'Network intrusion detection algorithm based on deep neural network', *IET Information Security*, 13(1), pp. 48–53. Available at: https://doi.org/https://doi.org/10.1049/iet-ifs.2018.5258.

Jiang, F. *et al.* (2020) 'Deep Learning Based Multi-Channel Intelligent Attack Detection for Data Security', *IEEE Transactions on Sustainable Computing*, 5(2), pp. 204–212. Available at: https://doi.org/10.1109/TSUSC.2018.2793284.

Jianliang, M., Haikun, S. and Ling, B. (2009) 'The Application on Intrusion Detection Based on K-means Cluster Algorithm', in *2009 International Forum on Information Technology and Applications*, pp. 150–152. Available at: https://doi.org/10.1109/IFITA.2009.34.

Jyothsna, Prasad, R. and Prasad, M. (2011) 'A Review of Anomaly based Intrusion Detection Systems', *International journal of computer applications*, 28(7), pp. 26–35. Available at: https://doi.org/10.5120/3399-4730.

Kang, H. *et al.* (2019) 'IoT network intrusion dataset', *IEEE Dataport* [Preprint]. Available at:

https://dx.doi.org/10.21227/q70p-q449 (Accessed: 9 August 2022).

'KDD dataset' (1999). Available at: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99. html (Accessed: 9 August 2022).

Khan, I.A. *et al.* (2022) 'XSRU-IoMT: Explainable simple recurrent units for threat detection in Internet of Medical Things networks', *Future Generation Computer Systems*, 127, pp. 181–193. Available at: https://doi.org/10.1016/j.future.2021.09.010.

Khraisat, A. and Alazab, A. (2021) 'A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges', *Cybersecurity*, 4(1), p. 18. Available at: https://doi.org/10.1186/s42400-021-00077-7.

Kim, D.S., Nguyen, H.-N. and Park, J.S. (2005) 'Genetic algorithm to improve SVM based network intrusion detection system', in *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, pp. 155–158 vol.2. Available at: https://doi.org/10.1109/AINA.2005.191.

Koc, L., Mazzuchi, T.A. and Sarkani, S. (2012) 'A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier', *Expert Systems with Applications*, 39(18), pp. 13492–13500. Available at: https://doi.org/https://doi.org/10.1016/j.eswa.2012.07.009.

Koroniotis, N. *et al.* (2019) 'Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset', *Future Generation Computer Systems*, 100, pp. 779–796. Available at: https://doi.org/https://doi.org/10.1016/j.future.2019.05.041.

Kreibich, C. and Crowcroft, J. (2004) 'Honeycomb: Creating Intrusion Detection Signatures Using Honeypots', *SIGCOMM Comput. Commun. Rev.*, 34(1), pp. 51–56. Available at: https://doi.org/10.1145/972374.972384.

Krüger, F. (2016) *Activity, Context, and Plan Recognition with Computational Causal Behaviour Models*. Available at: https://www.researchgate.net/publication/314116591.

Latif, S. *et al.* (2021) 'Intrusion Detection Framework for the Internet of Things using a Dense Random Neural Network', *IEEE Transactions on Industrial Informatics* [Preprint]. Available at: https://doi.org/10.1109/TII.2021.3130248.

Lin, T.-Y. *et al.* (2017) 'Focal Loss for Dense Object Detection'. Available at: http://arxiv.org/abs/1708.02002.

Lo, W.W. *et al.* (2022) 'E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT', in. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–9. Available at: https://doi.org/10.1109/noms54207.2022.9789878.

Longadge, M.R. *et al.* (2013) *Class Imbalance Problem in Data Mining: Review, International Journal of Computer Science and Network*. Available at: www.ijcsn.org.

Masdari, M. and Khezri, H. (2020) 'A survey and taxonomy of the fuzzy signature-based Intrusion Detection Systems', *Applied Soft Computing*, 92, p. 106301. Available at: https://doi.org/https://doi.org/10.1016/j.asoc.2020.106301.

Meidan, Y. *et al.* (2018) 'N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders', *IEEE Pervasive Computing*, 17(3), pp. 12–22. Available at: https://doi.org/10.1109/MPRV.2018.03367731.
Mohamed, A., Dahl, G. and Hinton, G. (2009) 'Deep Belief Networks for phone recognition', 1(9).

el Naqa Issam and Murphy, M.J. (2015) 'What Is Machine Learning?', in R. and M.M.J. el Naqa Issam and Li (ed.) *Machine Learning in Radiation Oncology: Theory and Applications*. Cham: Springer International Publishing, pp. 3–11. Available at: https://doi.org/10.1007/978-3-319-18305-3_1.

O'Shea, K. and Nash, R. (2015) 'An Introduction to Convolutional Neural Networks', *arXiv preprint arXiv:1511.08458* [Preprint].

Panagiotou, P. *et al.* (2021) 'Host-based Intrusion Detection Using Signature-based and AI-driven Anomaly Detection Methods', *Information & Security: An International Journal*, 50, pp. 37–48. Available at: https://doi.org/10.11610/isij.5016.

Panda, M. and Ranjan Patra, M. (2007) 'NETWORK INTRUSION DETECTION USING NAÏVE BAYES', *IJCSNS International Journal of Computer Science and Network Security*, 7(12).

Panja, B., Ogunyanwo, O. and Meharia, P. (2014) 'Training of intelligent intrusion detection system using neuro fuzzy', in *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1–6. Available at: https://doi.org/10.1109/SNPD.2014.6888688.

Peng, K., Leung, V.C.M. and Huang, Q. (2018) 'Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System Over Big Data', *IEEE Access*, 6, pp. 11897–11906. Available at: https://doi.org/10.1109/ACCESS.2018.2810267.

Peterson, J.M., Leevy, J.L. and Khoshgoftaar, T.M. (2021) 'A Review and Analysis of the Bot-IoT Dataset', in *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 20–27. Available at: https://doi.org/10.1109/SOSE52839.2021.00007.

Petrov, C. (2022) '49 Stunning Internet of Things Statistics 2022 [The Rise of IoT]'. Available at: https://techjury.net/blog/internet-of-things-statistics/ (Accessed: 12 September 2022).

Polikar, R. (2006) 'Ensemble based systems in decision making', *IEEE Circuits and Systems Magazine*, 6(3), pp. 21–45. Available at: https://doi.org/10.1109/MCAS.2006.1688199.

Potluri, S. and Diedrich, C. (2016) 'Accelerated deep neural networks for enhanced Intrusion Detection System', in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8. Available at: https://doi.org/10.1109/ETFA.2016.7733515.

Rai, K., Devi, S. and Guleria, A. (2016) 'Decision Tree Based Algorithm for Intrusion Detection', *Int. J. Advanced Networking and Applications*, 07(04), pp. 2828–2834.

Roesch, M. (1999) *Snort-Lightweight Intrusion Detection for Networks*.

Ryan, J., Lin, M.-J. and Miikkulainen, R. (1997) 'Intrusion Detection with Neural Networks', in M. Jordan, M. Kearns, and S. Solla (eds) *Advances in Neural Information Processing Systems*. MIT Press. Available at: https://proceedings.neurips.cc/paper/1997/file/1abb1e1ea5f481b589da52303b091cbb-Paper.pdf.

Seo, J. and Lee, S. (2016) 'A study on efficient detection of network-based IP spoofing DDoS and malware-infected Systems', *SpringerPlus*, 5. Available at: https://doi.org/10.1186/s40064-016-3569-3.

Shah, B. and Trivedi, B.H. (2012) 'Artificial Neural Network based Intrusion Detection System: A Survey', *International Journal of Computer Applications*, 39(6).

Sherasiya, T., Upadhyay, H. and Patel, H.B. (2016) *A SURVEY: INTRUSION DETECTION SYSTEM FOR INTERNET OF THINGS*. Available at: www.iaset.us.

Sivanathan, A. *et al.* (2019) 'Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics', *IEEE Transactions on Mobile Computing*, 18(8), pp. 1745–1759. Available at: https://doi.org/10.1109/TMC.2018.2866249.

Sivatha Sindhu, S.S., Geetha, S. and Kannan, A. (2012) 'Decision tree based light weight intrusion detection using a wrapper approach', *Expert Systems with Applications*, 39(1), pp. 129–141. Available at: https://doi.org/https://doi.org/10.1016/j.eswa.2011.06.013.

Tama, B.A. and Lim, S. (2021) 'Ensemble learning for intrusion detection systems: A systematic mapping study and cross-benchmark evaluation', *Computer Science Review*, 39, p. 100357. Available at: https://doi.org/https://doi.org/10.1016/j.cosrev.2020.100357.

Tavallaee, M. *et al.* (2009) 'A detailed analysis of the KDD CUP 99 data set', in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–6. Available at: https://doi.org/10.1109/CISDA.2009.5356528.

Tawalbeh, L. *et al.* (2020) 'IoT Privacy and Security: Challenges and Solutions', *Applied Sciences*, 10(12). Available at: https://doi.org/10.3390/app10124102.

Thales Group (2021) 'IOT SECURITY ISSUES IN 2022: A BUSINESS PERSPECTIVE'. Available at: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/magazine/internet-threats (Accessed: 12 September 2022).

Waagsnes, H. and Ulltveit-Moe, N. (2018) 'Intrusion detection system test framework for SCADA systems', in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*. SciTePress, pp. 275–285. Available at: https://doi.org/10.5220/0006588202750285.

Wang, G. *et al.* (2010) 'A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering', *Expert Systems with Applications*, 37(9), pp. 6225–6232. Available at: https://doi.org/https://doi.org/10.1016/j.eswa.2010.02.102.

Wei, P. *et al.* (2019) 'An Optimization Method for Intrusion Detection Classification Model Based on Deep Belief Network', *IEEE Access*, 7, pp. 87593–87605. Available at: https://doi.org/10.1109/ACCESS.2019.2925828.

Xiao, Y. *et al.* (2019) 'An Intrusion Detection Model Based on Feature Reduction and Convolutional Neural Networks', *IEEE Access*, 7, pp. 42210–42219. Available at: https://doi.org/10.1109/ACCESS.2019.2904620.

Xu, C. *et al.* (2018) 'An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units', *IEEE Access*, 6, pp. 48697–48707. Available at: https://doi.org/10.1109/ACCESS.2018.2867564.

Yan, B. and Han, G. (2018) 'Effective Feature Extraction via Stacked Sparse Autoencoder to Improve Intrusion Detection System', *IEEE Access*, 6, pp. 41238–41248. Available at: https://doi.org/10.1109/ACCESS.2018.2858277.

Yang, H., Cheng, L. and Chuah, M.C. (2019) 'Deep-Learning-Based Network Intrusion Detection for SCADA Systems', in *2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–7. Available at: https://doi.org/10.1109/CNS.2019.8802785.

Yin, C. *et al.* (2017) 'A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks', *IEEE Access*, 5, pp. 21954–21961. Available at: https://doi.org/10.1109/ACCESS.2017.2762418.

Yu, L. *et al.* (2022) 'Missing Data Preprocessing in Credit Classification: One-Hot Encoding or Imputation?', *Emerging Markets Finance and Trade*, 58(2), pp. 472–482. Available at: https://doi.org/10.1080/1540496X.2020.1825935.

Zeidanloo, H.R. *et al.* (2010) 'A taxonomy of Botnet detection techniques', in *2010 3rd International Conference on Computer Science and Information Technology*, pp. 158–162. Available at: https://doi.org/10.1109/ICCSIT.2010.5563555.

Zhang, W., Li, X. and Ding, Q. (2019) 'Deep residual learning-based fault diagnosis method for rotating machinery', *ISA Transactions*, 95, pp. 295–305. Available at: https://doi.org/10.1016/j.isatra.2018.12.025.

Zhang, X. *et al.* (2019) 'A Multiple-Layer Representation Learning Model for Network-Based Attack Detection', *IEEE Access*, 7, pp. 91992–92008. Available at: https://doi.org/10.1109/ACCESS.2019.2927465.

8 APPENDIX I

The links and files related to this project are provided in this section.

- The TON_IoT dataset used in this study can be accessed at Train_Test_Network_dataset - Files - CloudStor (aarnet.edu.au).

- The project is implemented in Jupyter Notebook which was developed in Google Collab. The project can be accessed in Google Collab through: https://colab.research.google.com/drive/1q6Tii74E-h3tPRI3Vim2hxkUq-SZpIWW?usp=sharing

The details of the meeting with supervisor are provided below.

Date: 19th May 2022

Mode: Online through MS Teams

Details: Discussed about my previous submission and then my supervisor gave me his feedback on idea of using TON_IoT dataset for Intrusion detection.

9 APPENDIX II

I. The description of the network features of the TON_IoT dataset is provided below which can be accessed at [Description_stats_Network_dataset - Files - CloudStor (aarnet.edu.au)]

Description of Network Features

Service profile: Connection activity

ID	Feature	Type	Description
1	ts	Time	Timestamp of connection between flow identifiers
2	src_ip	String	Source IP addresses which originate endpoints' IP addresses
3	src_port	Number	Source ports which Originate endpoint's TCP/UDP ports
4	dst_ip	String	Destination IP addresses which respond to endpoint's IP addresses
5	dst_port	Number	Destination ports which respond to endpoint's TCP/UDP ports
6	proto	String	Transport layer protocols of flow connections
7	service	String	Dynamically detected protocols, such as DNS, HTTP and SSL
8	duration	Number	The time of the packet connections, which is estimated by subtracting 'time of last packet seen' and 'time of first packet seen'
9	src_bytes	Number	Source bytes which are originated from payload bytes of TCP sequence numbers
10	dst_bytes	Number	Destination bytes which are responded payload bytes from TCP sequence numbers
11	conn_state	String	Various connection states, such as S0 (connection without replay), S1 (connection established), and REJ (connection attempt rejected)
12	missed_bytes	Number	Number of missing bytes in content gaps

Service profile: Statistical activity

ID	Feature	Type	Description
13	src_pkts	Number	Number of original packets which is estimated from source systems
14	src_ip_bytes	Number	Number of original IP bytes which is the total length of IP header field of source systems
15	dst_pkts	Number	Number of destination packets which is estimated from destination systems
16	dst_ip_bytes	Number	Number of destination IP bytes which is the total length of IP header field of destination systems

Service profile: DNS activity

ID	Feature	Type	Description
17	dns_query	string	Domain name subjects of the DNS queries
18	dns_qclass	Number	Values which specifies the DNS query classes
19	dns_qtype	Number	Value which specifies the DNS query types
20	dns_rcode	Number	Response code values in the DNS responses
21	dns_AA	Bool	Authoritative answers of DNS, where T denotes server is authoritative for query
22	dns_RD	Bool	Recursion desired of DNS, where T denotes request recursive lookup of query
23	dns_RA	Bool	Recursion available of DNS, where T denotes server supports recursive queries
24	dns_rejected	Bool	DNS rejection, where the DNS queries are rejected by the server

Service profile: SSL activity

ID	Feature	Type	Description
25	ssl_version	String	SSL version which is offered by the server
26	ssl_cipher	String	SSL cipher suite which the server chose
27	ssl_resumed	Bool	SSL flag indicates the session that can be used to initiate new connections, where T refers to the SSL connection is initiated
28	ssl_established	Bool	SSL flag indicates establishing connections between two parties,

			where T refers to establishing the connection
29	ssl_subject	String	Subject of the X.509 cert offered by the server
30	ssl_issuer	String	Trusted owner/originator of SLL and digital certificate (certificate authority)

Service profile: HTTP activity

ID	Feature	Type	Description
31	http_trans_depth	Number	Pipelined depth into the HTTP connection
32	http_method	String	HTTP request methods such as GET, POST and HEAD
33	http_uri	String	URIs used in the HTTP request
35	http_version	String	The HTTP versions utilised such as V1.1
36	http_request_body_len	Number	Actual uncompressed content sizes of the data transferred from the HTTP client
37	http_response_body_len	Number	Actual uncompressed content sizes of the data transferred from the HTTP server
38	http_status_code	Number	Status codes returned by the HTTP server
39	http_user_agent	Number	Values of the User-Agent header in the HTTP protocol
40	http_orig_mime_types	String	Ordered vectors of mime types from source system in the HTTP protocol
41	http_resp_mime_types	String	Ordered vectors of mime types from destination system in the HTTP protocol

Service profile: Violation activity

ID	Feature	Type	Description
42	weird_name	String	Names of anomalies/violations related to protocols that happened
43	weird_addl	String	Additional information is associated to protocol anomalies/violations
44	weird_notice	bool	It indicates if the violation/anomaly was turned into a notice

Service profile: Data labelling

ID	Feature	Type	Description
45	label	Number	Tag normal and attack records, where 0 indicates normal and 1 indicates attacks
46	type	String	Tag attack categories, such as normal, DoS, DDoS and backdoor attacks, and normal records

II. The experimental setup of the proposed system in this research work is provided below

A. Hardware

The models built were trained on a Windows 10 workstation and on Google Cloud.

1) Workstation:

a. Processor: AMD Ryzen 5 2600 Six-Core Processor 3.40 GHz

b. RAM: 16.0 GB

c. System Type: 64-bit OS, x-64 based processor

2) Google Cloud:

a. Processor: Python 3 Google Compute Engine Backend

b. RAM: 12.7 GB

B. Software and Libraries

1) Jupyter Notebook: It is an open-source application that facilitates data preprocessing, statistical modeling, data visualization, machine learning and much more.

2) Google Colab: It is a colab notebook hosted on google cloud servers providing

access to GPU's and TPU's for tasks that can be done in a Jupyter notebook

3) Kaggle: It allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers.

4) Libraries: Python was used as the scripting language for writing most of the code.

a. Scikit-learn: It is used for predictive analytics tasks like classification, regression, clustering, dimensionality reduction, model selection and preprocessing.

b. Imblearn: This library provides an API for imbalanced learning and wide samples.

c. Xgboost: It is a highly efficient gradient boosting library for xgboost classifiers.

d. Pandas: It is a data analysis and manipulation library implemented in python

e. NumPy: It provides numerical computing capability and high-level mathematical functions for multidimensional arrays and matrices.

f. Matplotlib & seaborn: This package was used to create visualizations.

III. The experimental code for this research is provided below.

Intrusion Detection using Deep learning approaches

Importing Dataset

```
from google.colab import files
files.upload()
```

```
Mounted at /content/drive
```

```
{"type":"string"}
```

Pre-processing the Dataset

```
# Import required packages
```

```
import numpy as np  # linear algebra
import pandas as pd # Data processing, CSV file I/O (e.g. pd.read_csv)
import os
import tensorflow as tf
%matplotlib inline
import matplotlib.pyplot as plt
from glob import glob
import seaborn as sns
from PIL import Image
import random
import pickle
import cv2
import datetime
from pprint import pprint
import librosa
from itertools import cycle


from sklearn.model_selection import *
from sklearn.preprocessing import *
from sklearn.metrics import *
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_recall_fscore_support,  roc_curve, auc

import keras
from keras.utils import np_utils
from keras.utils.vis_utils import plot_model
from keras.regularizers import *
from keras.initializers import glorot_uniform

import keras.backend as K
K.clear_session()

from keras.models import *
from keras.layers import *
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import *
from keras.callbacks import *



data=pd.read_csv('/content/drive/My Drive/Train_Test_Network.csv')

data
```

```
                ts         src_ip  src_port         dst_ip  dst_port proto  \
0       1554198358    3.122.49.24      1883  192.168.1.152     52976   tcp
1       1554198358   192.168.1.79     47260  192.168.1.255     15600   udp
2       1554198359  192.168.1.152      1880  192.168.1.152     51782   tcp
3       1554198359  192.168.1.152     34296  192.168.1.152     10502   tcp
4       1554198362  192.168.1.152     46608  192.168.1.190        53   udp
```

```
...           ...            ...       ...             ...       ...   ...
461038  1556340862   192.168.1.32     33108  176.28.50.165        80   tcp
461039  1556423390   192.168.1.33     37242  34.230.157.88       443   tcp
461040  1556436603   192.168.1.37      4444  192.168.1.193     49178   tcp
461041  1556540442   192.168.1.31     60816  104.98.29.100       443   tcp
461042  1556540442   192.168.1.31     41054  151.101.24.64       443   tcp

       service      duration  src_bytes  dst_bytes  ...  \
0            -  80549.530260    1762852   41933215  ...
1            -      0.000000          0          0  ...
2            -      0.000000          0          0  ...
3            -      0.000000          0          0  ...
4          dns      0.000549          0        298  ...
...        ...           ...        ...        ...  ...
461038       -      0.000000          0          0  ...
461039       -      0.000000          0          0  ...
461040       -    290.371539     101568       2592  ...
461041       -     23.190902         32         31  ...
461042       -     22.872314         32         31  ...

       http_response_body_len  http_status_code  http_user_agent  \
0                           0                 0                -
1                           0                 0                -
2                           0                 0                -
3                           0                 0                -
4                           0                 0                -
...                       ...               ...              ...
461038                      0                 0                -
461039                      0                 0                -
461040                      0                 0                -
461041                      0                 0                -
461042                      0                 0                -

        http_orig_mime_types  http_resp_mime_types        weird_name  \
0                          -                     -  bad_TCP_checksum
1                          -                     -                 -
2                          -                     -  bad_TCP_checksum
3                          -                     -                 -
4                          -                     -  bad_UDP_checksum
...                      ...                   ...               ...
461038                     -                     -                 -
461039                     -                     -                 -
461040                     -                     -                 -
461041                     -                     -                 -
461042                     -                     -                 -

       weird_addl  weird_notice  label        type
0               -             F      0      normal
1               -             -      0      normal
2               -             F      0      normal
3               -             -      0      normal
```

```
4               -             F      0      normal
...           ...           ...    ...         ...
461038          -             -      1         xss
461039          -             -      1  ransomware
461040          -             -      1    backdoor
461041          -             -      1        mitm
461042          -             -      1        mitm

[461043 rows x 45 columns]
```

 # Shuffling the rows of dataframe

```
sampler=np.random.permutation(len(data))

data=data.take(sampler)
print(data.shape)
```

```
(461043, 45)
```

 # Dummy encode labels and stored separately

```
labels_full=pd.get_dummies(data['type'], prefix='type')
labels_full.head()
```

```
        type_backdoor  type_ddos  type_dos  type_injection  type_mitm  \
235902              0          0         0               0          0
324566              0          0         0               0          0
386433              0          0         0               0          0
19885               0          0         0               0          0
416651              0          0         0               0          0

        type_normal  type_password  type_ransomware  type_scanning  type_xss
235902            1              0                0              0         0
324566            0              1                0              0         0
386433            1              0                0              0         0
19885             1              0                0              0         0
416651            1              0                0              0         0
```

 # Extracting required features from the Dataset

```
data=data[['ts', 'src_port', 'dst_port', 'proto', 'service', 'duration',
'src_bytes', 'dst_bytes', 'conn_state', 'missed_bytes', 'src_pkts',
'src_ip_bytes', 'dst_pkts', 'dst_ip_bytes']]
data.head()
```

```
                ts  src_port  dst_port proto service  duration  src_bytes  \
235902  1556248537     58054        53   udp     dns  0.000758          0
324566  1556287572     60780        80   tcp       -  0.000142          0
386433  1556349852     61599      1900   udp       -  3.356777       3016
19885   1554226121     42100      7878   tcp       -  0.000000          0
416651  1556423583      9197     49822   tcp       -  0.000000          0
```

```
        dst_bytes conn_state  missed_bytes  src_pkts  src_ip_bytes  dst_pkts  \
235902        566        SHR             0         0             0         2
324566          0        REJ             0         1            60         1
386433          0         S0             0        24          3688         0
19885           0        OTH             0         0             0         0
416651          0        OTH             0         1            40         0

        dst_ip_bytes
235902           622
324566            40
386433             0
19885              0
416651             0
```

data1=pd.get_dummies(data=data, columns=['conn_state','proto', 'service'])

data1.head()

```
                ts  src_port  dst_port  duration  src_bytes  dst_bytes  \
235902  1556248537     58054        53  0.000758          0        566
324566  1556287572     60780        80  0.000142          0          0
386433  1556349852     61599      1900  3.356777       3016          0
19885   1554226121     42100      7878  0.000000          0          0
416651  1556423583      9197     49822  0.000000          0          0

        missed_bytes  src_pkts  src_ip_bytes  dst_pkts  ...  service_-  \
235902             0         0             0         2  ...          0
324566             0         1            60         1  ...          1
386433             0        24          3688         0  ...          1
19885              0         0             0         0  ...          1
416651             0         1            40         0  ...          1

        service_dce_rpc  service_dhcp  service_dns  service_ftp  \
235902                0             0            1            0
324566                0             0            0            0
386433                0             0            0            0
19885                 0             0            0            0
416651                0             0            0            0

        service_gssapi  service_http  service_smb  service_smb;gssapi  \
235902               0             0            0                   0
324566               0             0            0                   0
386433               0             0            0                   0
19885                0             0            0                   0
416651               0             0            0                   0

        service_ssl
235902            0
324566            0
386433            0
```

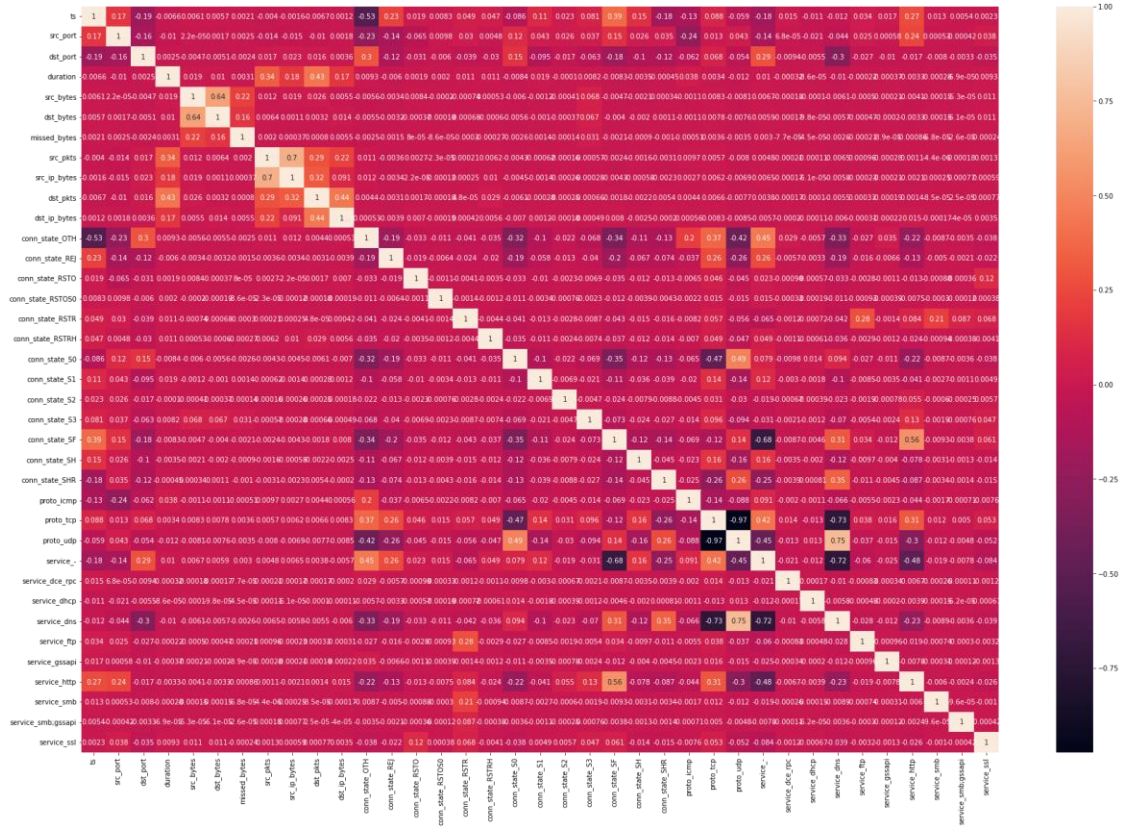
```
19885             0
416651            0

[5 rows x 37 columns]
```

```
# Data Standardization

def standardize(df,col):
    df[col]= (df[col]-df[col].mean())/df[col].std()


data_st=data1.copy()
for i in (data_st.iloc[:,].columns):
    standardize (data_st,i)

data_st.head()
```

```
              ts  src_port  dst_port  duration  src_bytes  dst_bytes  \
235902  0.660635  1.013977 -0.552686 -0.018484  -0.010455  -0.009767
324566  0.701901  1.161572 -0.550750 -0.018485  -0.010455  -0.009813
386433  0.767742  1.205916 -0.420224 -0.011163  -0.010194  -0.009813
19885  -1.477399  0.150173  0.008502 -0.018486  -0.010455  -0.009813
416651  0.845688 -1.631309  3.016615 -0.018486  -0.010455  -0.009813

        missed_bytes  src_pkts  src_ip_bytes  dst_pkts  ...  service_-  \
235902     -0.004464 -0.013855     -0.010581 -0.008149  ...  -1.244842
324566     -0.004464 -0.011924     -0.010232 -0.009436  ...   0.803313
386433     -0.004464  0.032482      0.010855 -0.010723  ...   0.803313
19885      -0.004464 -0.013855     -0.010581 -0.010723  ...   0.803313
416651     -0.004464 -0.011924     -0.010349 -0.010723  ...   0.803313

        service_dce_rpc  service_dhcp  service_dns  service_ftp  \
235902        -0.017178     -0.009989     1.719920    -0.048118
324566        -0.017178     -0.009989    -0.581421    -0.048118
386433        -0.017178     -0.009989    -0.581421    -0.048118
19885         -0.017178     -0.009989    -0.581421    -0.048118
416651        -0.017178     -0.009989    -0.581421    -0.048118

        service_gssapi  service_http  service_smb  service_smb;gssapi  \
235902       -0.019981     -0.389458    -0.015307           -0.006248
324566       -0.019981     -0.389458    -0.015307           -0.006248
386433       -0.019981     -0.389458    -0.015307           -0.006248
19885        -0.019981     -0.389458    -0.015307           -0.006248
416651       -0.019981     -0.389458    -0.015307           -0.006248

        service_ssl
235902    -0.067157
324566    -0.067157
386433    -0.067157
19885     -0.067157
```

```
416651    -0.067157

[5 rows x 37 columns]
```

```
# Feature Selection

def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):


        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in
absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr


#Using Pearson Correlation
plt.figure(figsize=(30,20))
cor = data_st.corr()
sns.heatmap(cor, annot=True)
plt.show()
```



```
corr_features = correlation(data_st, 0.9)
data_st = data_st.drop(corr_features,axis=1)

# Focal Loss for Class Imbalance

# Compatible with tensorflow backend

def focal_loss(gamma=2., alpha=.25):
      def focal_loss_fixed(y_true, y_pred):
            pt_1 = tf.where(tf.equal(y_true, 1), y_pred, tf.ones_like(y_pred))
            pt_0 = tf.where(tf.equal(y_true, 0), y_pred, tf.zeros_like(y_pred))
            return -K.mean(alpha * K.pow(1. - pt_1, gamma) *
K.log(pt_1+K.epsilon())) - K.mean((1 - alpha) * K.pow(pt_0, gamma) * K.log(1. -
pt_0 + K.epsilon()))
      return focal_loss_fixed

train_data_st=data_st.values
train_data_st

array([[ 0.66063487,  1.01397712, -0.55268595, ..., -0.01530705,
        -0.00624847, -0.06715698],
       [ 0.70190142,  1.16157217, -0.55074958, ..., -0.01530705,
        -0.00624847, -0.06715698],
       [ 0.76774185,  1.20591565, -0.42022399, ..., -0.01530705,
        -0.00624847, -0.06715698],
       ...,
       [-1.41441263, -0.06737584,  0.56230384, ..., -0.01530705,
        -0.00624847, -0.06715698],
       [ 0.77484602,  1.16270918, -0.17244051, ..., -0.01530705,
        -0.00624847, -0.06715698],
       [-1.43484875,  1.08560891,  0.56230384, ..., -0.01530705,
        -0.00624847, -0.06715698]])

# Labels for training

labels=labels_full.values
labels

array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)

# Validation Technique

# Splitting the Data for Training(70%) and Testing(30%) the model
x_train, x_test, y_train, y_test = train_test_split(train_data_st, labels,
test_size=0.3)
```

```
# Splitting the Training Data for Training(87.5%) and Validating(12.5%) the model
x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train,
test_size=0.125)

print(x_train.shape)
print(x_test.shape)
print(x_validate.shape)
```

```
(282388, 36)
(138313, 36)
(40342, 36)
```

Neural Network Models

CNN-LSTM Model

```
model1 = Sequential()
model1.add(Conv1D(filters=64, kernel_size=5, strides=1, padding='same',
input_shape = (train_data_st.shape[1], 1)))
model1.add(Conv1D(filters=32, kernel_size=5, strides=1, padding='same'))
model1.add(LSTM(32, activation = 'relu', return_sequences=True))
model1.add(LSTM(16, return_sequences=True))  # returns a sequence of vectors of
dimension 16
model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(Dense(64, activation='relu'))

model1.add(Dense(labels.shape[1],activation='softmax'))

model1Name = 'CNN+LSTM'
keras.utils.vis_utils.plot_model(model1,
'./'+model1Name+'_Archi.png',show_shapes=True)
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 36, 64)            384

 conv1d_1 (Conv1D)           (None, 36, 32)            10272

 lstm (LSTM)                 (None, 36, 32)            8320

 lstm_1 (LSTM)               (None, 36, 16)            3136

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 128)               73856

 dense_1 (Dense)             (None, 64)                8256
```

```
 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 104,874
Trainable params: 104,874
Non-trainable params: 0
_________________________________________________________________
```

DRCNN Model

```
# Building 5 block of residual Convolutional layer

inp = Input(shape=(train_data_st.shape[1], 1))
C = Conv1D(filters=64, kernel_size=5, strides=1)(inp)

C11 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(C)
A11 = Activation("relu")(C11)
C12 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(A11)
S11 = Add()([C12, C])
A12 = Activation("relu")(S11)
M11 = MaxPooling1D(pool_size=3, strides=2)(A12)


C21 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(M11)
A21 = Activation("relu")(C21)
C22 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(A21)
S21 = Add()([C22, M11])
A22 = Activation("relu")(S11)
M21 = MaxPooling1D(pool_size=3, strides=2)(A22)


C31 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(M21)
A31 = Activation("relu")(C31)
C32 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(A31)
S31 = Add()([C32, M21])
A32 = Activation("relu")(S31)
M31 = MaxPooling1D(pool_size=3, strides=2)(A32)


C41 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(M31)
A41 = Activation("relu")(C41)
C42 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(A41)
S41 = Add()([C42, M31])
A42 = Activation("relu")(S41)
M41 = MaxPooling1D(pool_size=3, strides=2)(A42)


C51 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(M41)
A51 = Activation("relu")(C51)
C52 = Conv1D(filters=64, kernel_size=5, strides=1, padding='same')(A51)
S51 = Add()([C52, M41])
```

```
A52 = Activation("relu")(S51)
M51 = MaxPooling1D(pool_size=3, strides=2)(A52)

F1 = Flatten()(M51) #m51

D1 = Dense(32)(F1)
A6 = Activation("relu")(D1)
D2 = Dense(32)(A6)
D3 = Dense(labels.shape[1])(D2)
A7 = Activation("softmax")(D3)

model2 = Model(inputs=inp, outputs=A7)


keras.utils.vis_utils.plot_model(model2, './Deep_residual_CNN_model.png',
show_shapes=True)

model2Name='Deep residual CNN'

model2.summary()
```

```
Model: "model"
__________________________________________________________________________________
_________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
================
 input_1 (InputLayer)           [(None, 36, 1)]      0           []

 conv1d_2 (Conv1D)              (None, 32, 64)       384         ['input_1[0][0]']

 conv1d_3 (Conv1D)              (None, 32, 64)       20544
['conv1d_2[0][0]']

 activation (Activation)        (None, 32, 64)       0
['conv1d_3[0][0]']

 conv1d_4 (Conv1D)              (None, 32, 64)       20544
['activation[0][0]']

 add (Add)                      (None, 32, 64)       0
['conv1d_4[0][0]',

'conv1d_2[0][0]']

 activation_3 (Activation)      (None, 32, 64)       0           ['add[0][0]']

 max_pooling1d_1 (MaxPooling1D)  (None, 15, 64)      0
['activation_3[0][0]']
```

```
 conv1d_7 (Conv1D)              (None, 15, 64)       20544
['max_pooling1d_1[0][0]']

 activation_4 (Activation)      (None, 15, 64)       0
['conv1d_7[0][0]']

 conv1d_8 (Conv1D)              (None, 15, 64)       20544
['activation_4[0][0]']

 add_2 (Add)                    (None, 15, 64)       0
['conv1d_8[0][0]',

'max_pooling1d_1[0][0]']

 activation_5 (Activation)      (None, 15, 64)       0           ['add_2[0][0]']

 max_pooling1d_2 (MaxPooling1D)  (None, 7, 64)       0
['activation_5[0][0]']

 conv1d_9 (Conv1D)              (None, 7, 64)        20544
['max_pooling1d_2[0][0]']

 activation_6 (Activation)      (None, 7, 64)        0
['conv1d_9[0][0]']

 conv1d_10 (Conv1D)             (None, 7, 64)        20544
['activation_6[0][0]']

 add_3 (Add)                    (None, 7, 64)        0
['conv1d_10[0][0]',

'max_pooling1d_2[0][0]']

 activation_7 (Activation)      (None, 7, 64)        0           ['add_3[0][0]']

 max_pooling1d_3 (MaxPooling1D)  (None, 3, 64)       0
['activation_7[0][0]']

 conv1d_11 (Conv1D)             (None, 3, 64)        20544
['max_pooling1d_3[0][0]']

 activation_8 (Activation)      (None, 3, 64)        0
['conv1d_11[0][0]']

 conv1d_12 (Conv1D)             (None, 3, 64)        20544
['activation_8[0][0]']

 add_4 (Add)                    (None, 3, 64)        0
['conv1d_12[0][0]',
```

```
'max_pooling1d_3[0][0]']

 activation_9 (Activation)      (None, 3, 64)        0           ['add_4[0][0]']

 max_pooling1d_4 (MaxPooling1D)  (None, 1, 64)       0
['activation_9[0][0]']

 flatten_1 (Flatten)            (None, 64)           0
['max_pooling1d_4[0][0]']

 dense_3 (Dense)                (None, 32)           2080
['flatten_1[0][0]']

 activation_10 (Activation)     (None, 32)           0           ['dense_3[0][0]']

 dense_4 (Dense)                (None, 32)           1056
['activation_10[0][0]']

 dense_5 (Dense)                (None, 10)           330         ['dense_4[0][0]']

 activation_11 (Activation)     (None, 10)           0           ['dense_5[0][0]']

==================================================================================
================
Total params: 168,202
Trainable params: 168,202
Non-trainable params: 0
__________________________________________________________________________________
_______________
```

Training the Models

```
# Function to compile and fit the model


def model_val(model, modelName):
    adam = tf.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, decay=0.0)

    model.compile(loss=[focal_loss(alpha=.25, gamma=2)], optimizer=adam,
metrics=['accuracy'])
    learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
patience=3, verbose=1,
                                            factor=0.5,
                                            learning_rate=0.00001)

    earlystop = EarlyStopping(monitor = 'val_loss',
                              min_delta = 0,
                              patience = 10,
                              verbose = 1,
                              restore_best_weights = True)
```

```
    checkpoint = ModelCheckpoint('./'+modelName+'.h5',
                                 monitor='val_loss',
                                 mode='min',
                                 save_best_only=True,
                                 save_weights_only=True,
                                 verbose=1)

    history = model.fit(x_train,y_train, batch_size=batch_size,
    steps_per_epoch=x_train.shape[0] // batch_size,
    epochs=epochs,
    validation_data=(x_validate,y_validate),
    callbacks=[learning_rate_reduction, checkpoint]
    )


    plot_model_history(history, modelName)
    with open('./History_'+modelName, 'wb') as file_pi:
        pickle.dump(history.history, file_pi)

# Function to plot model's validation loss and validation accuracy

def plot_model_history(model_history, modelName):
    fig, axs = plt.subplots(1,2,figsize=(15,5))

    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),
model_history.history['accuracy'], '--*', color = (1,0,0))
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),
model_history.history['val_accuracy'], '-^', color = (0.7,0,0.7))
    axs[0].set_title('Model '+modelName+' Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')

axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_
history.history['accuracy'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    axs[0].grid('on')

    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),
model_history.history['loss'],'-x', color = (0,0.5,0))
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),
model_history.history['val_loss'],  '-.D', color = (0,0,0.5))
    axs[1].set_title('Model '+modelName+' Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')

axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_hist
ory.history['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
```

```
    axs[1].grid('on')
    plt.savefig('./'+modelName+'.jpg',dpi=600, quality = 100, optimize = True)
    plt.show()

epochs = 70
batch_size = 512

# Training CNN-LSTM model

model_val(model1, model1Name)
```

```
Epoch 1/70
551/551 [==============================] - ETA: 0s - loss: 0.0084 - accuracy:
0.8643
Epoch 1: val_loss improved from inf to 0.00533, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 83s 145ms/step - loss: 0.0084 -
accuracy: 0.8643 - val_loss: 0.0053 - val_accuracy: 0.9236 - lr: 0.0010
Epoch 2/70
551/551 [==============================] - ETA: 0s - loss: 0.0025 - accuracy:
0.9626
Epoch 2: val_loss improved from 0.00533 to 0.00104, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 83s 151ms/step - loss: 0.0025 -
accuracy: 0.9626 - val_loss: 0.0010 - val_accuracy: 0.9863 - lr: 0.0010
Epoch 3/70
551/551 [==============================] - ETA: 0s - loss: 9.4138e-04 - accuracy:
0.9853
Epoch 3: val_loss improved from 0.00104 to 0.00055, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 79s 144ms/step - loss: 9.4138e-04 -
accuracy: 0.9853 - val_loss: 5.5244e-04 - val_accuracy: 0.9908 - lr: 0.0010
Epoch 4/70
551/551 [==============================] - ETA: 0s - loss: 8.2951e-04 - accuracy:
0.9878
Epoch 4: val_loss improved from 0.00055 to 0.00055, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 78s 142ms/step - loss: 8.2951e-04 -
accuracy: 0.9878 - val_loss: 5.5062e-04 - val_accuracy: 0.9901 - lr: 0.0010
Epoch 5/70
551/551 [==============================] - ETA: 0s - loss: 0.0010 - accuracy:
0.9855
Epoch 5: val_loss improved from 0.00055 to 0.00047, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 78s 142ms/step - loss: 0.0010 -
accuracy: 0.9855 - val_loss: 4.6839e-04 - val_accuracy: 0.9937 - lr: 0.0010
Epoch 6/70
551/551 [==============================] - ETA: 0s - loss: 7.7148e-04 - accuracy:
0.9887
Epoch 6: val_loss did not improve from 0.00047
551/551 [==============================] - 78s 142ms/step - loss: 7.7148e-04 -
accuracy: 0.9887 - val_loss: 8.5513e-04 - val_accuracy: 0.9879 - lr: 0.0010
Epoch 7/70
551/551 [==============================] - ETA: 0s - loss: 4.7837e-04 - accuracy:
0.9926
```

```
Epoch 7: val_loss improved from 0.00047 to 0.00037, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 78s 142ms/step - loss: 4.7837e-04 -
accuracy: 0.9926 - val_loss: 3.7155e-04 - val_accuracy: 0.9937 - lr: 0.0010
Epoch 8/70
551/551 [==============================] - ETA: 0s - loss: 7.0987e-04 - accuracy:
0.9895
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 8: val_loss did not improve from 0.00037
551/551 [==============================] - 78s 142ms/step - loss: 7.0987e-04 -
accuracy: 0.9895 - val_loss: 0.0018 - val_accuracy: 0.9802 - lr: 0.0010
Epoch 9/70
551/551 [==============================] - ETA: 0s - loss: 4.8103e-04 - accuracy:
0.9928
Epoch 9: val_loss improved from 0.00037 to 0.00034, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 78s 142ms/step - loss: 4.8103e-04 -
accuracy: 0.9928 - val_loss: 3.4324e-04 - val_accuracy: 0.9940 - lr: 5.0000e-04
Epoch 10/70
551/551 [==============================] - ETA: 0s - loss: 3.4030e-04 - accuracy:
0.9945
Epoch 10: val_loss improved from 0.00034 to 0.00033, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 78s 142ms/step - loss: 3.4030e-04 -
accuracy: 0.9945 - val_loss: 3.2963e-04 - val_accuracy: 0.9944 - lr: 5.0000e-04
Epoch 11/70
551/551 [==============================] - ETA: 0s - loss: 3.7618e-04 - accuracy:
0.9937
Epoch 11: val_loss did not improve from 0.00033
551/551 [==============================] - 78s 142ms/step - loss: 3.7618e-04 -
accuracy: 0.9937 - val_loss: 9.6515e-04 - val_accuracy: 0.9854 - lr: 5.0000e-04
Epoch 12/70
551/551 [==============================] - ETA: 0s - loss: 0.0013 - accuracy:
0.9814
Epoch 12: val_loss did not improve from 0.00033
551/551 [==============================] - 78s 142ms/step - loss: 0.0013 -
accuracy: 0.9814 - val_loss: 0.0015 - val_accuracy: 0.9818 - lr: 5.0000e-04
Epoch 13/70
551/551 [==============================] - ETA: 0s - loss: 8.5193e-04 - accuracy:
0.9877
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 13: val_loss did not improve from 0.00033
551/551 [==============================] - 78s 142ms/step - loss: 8.5193e-04 -
accuracy: 0.9877 - val_loss: 5.3879e-04 - val_accuracy: 0.9919 - lr: 5.0000e-04
Epoch 14/70
551/551 [==============================] - ETA: 0s - loss: 4.6377e-04 - accuracy:
0.9929
Epoch 14: val_loss did not improve from 0.00033
551/551 [==============================] - 78s 142ms/step - loss: 4.6377e-04 -
accuracy: 0.9929 - val_loss: 4.1541e-04 - val_accuracy: 0.9931 - lr: 2.5000e-04
Epoch 15/70
551/551 [==============================] - ETA: 0s - loss: 4.4019e-04 - accuracy:
```

```
0.9931
Epoch 15: val_loss did not improve from 0.00033
551/551 [==============================] - 78s 142ms/step - loss: 4.4019e-04 -
accuracy: 0.9931 - val_loss: 3.7144e-04 - val_accuracy: 0.9935 - lr: 2.5000e-04
Epoch 16/70
551/551 [==============================] - ETA: 0s - loss: 3.6313e-04 - accuracy:
0.9940
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.

Epoch 16: val_loss did not improve from 0.00033
551/551 [==============================] - 78s 142ms/step - loss: 3.6313e-04 -
accuracy: 0.9940 - val_loss: 3.4464e-04 - val_accuracy: 0.9942 - lr: 2.5000e-04
Epoch 17/70
551/551 [==============================] - ETA: 0s - loss: 3.1621e-04 - accuracy:
0.9945
Epoch 17: val_loss improved from 0.00033 to 0.00031, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 147ms/step - loss: 3.1621e-04 -
accuracy: 0.9945 - val_loss: 3.0959e-04 - val_accuracy: 0.9948 - lr: 1.2500e-04
Epoch 18/70
551/551 [==============================] - ETA: 0s - loss: 3.2083e-04 - accuracy:
0.9944
Epoch 18: val_loss did not improve from 0.00031
551/551 [==============================] - 80s 145ms/step - loss: 3.2083e-04 -
accuracy: 0.9944 - val_loss: 4.0591e-04 - val_accuracy: 0.9930 - lr: 1.2500e-04
Epoch 19/70
551/551 [==============================] - ETA: 0s - loss: 3.0488e-04 - accuracy:
0.9948
Epoch 19: val_loss improved from 0.00031 to 0.00028, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 78s 142ms/step - loss: 3.0488e-04 -
accuracy: 0.9948 - val_loss: 2.8063e-04 - val_accuracy: 0.9945 - lr: 1.2500e-04
Epoch 20/70
551/551 [==============================] - ETA: 0s - loss: 3.8341e-04 - accuracy:
0.9934
Epoch 20: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

Epoch 20: val_loss did not improve from 0.00028
551/551 [==============================] - 78s 142ms/step - loss: 3.8341e-04 -
accuracy: 0.9934 - val_loss: 3.6838e-04 - val_accuracy: 0.9935 - lr: 1.2500e-04
Epoch 21/70
551/551 [==============================] - ETA: 0s - loss: 3.2218e-04 - accuracy:
0.9941
Epoch 21: val_loss did not improve from 0.00028
551/551 [==============================] - 78s 142ms/step - loss: 3.2218e-04 -
accuracy: 0.9941 - val_loss: 3.5240e-04 - val_accuracy: 0.9937 - lr: 6.2500e-05
Epoch 22/70
551/551 [==============================] - ETA: 0s - loss: 3.1202e-04 - accuracy:
0.9944
Epoch 22: val_loss did not improve from 0.00028
551/551 [==============================] - 78s 142ms/step - loss: 3.1202e-04 -
accuracy: 0.9944 - val_loss: 3.0141e-04 - val_accuracy: 0.9950 - lr: 6.2500e-05
Epoch 23/70
```

```
551/551 [==============================] - ETA: 0s - loss: 2.9124e-04 - accuracy:
0.9946
Epoch 23: val_loss did not improve from 0.00028
551/551 [==============================] - 79s 143ms/step - loss: 2.9124e-04 -
accuracy: 0.9946 - val_loss: 2.8356e-04 - val_accuracy: 0.9953 - lr: 6.2500e-05
Epoch 24/70
551/551 [==============================] - ETA: 0s - loss: 2.7703e-04 - accuracy:
0.9950
Epoch 24: val_loss did not improve from 0.00028
551/551 [==============================] - 79s 144ms/step - loss: 2.7703e-04 -
accuracy: 0.9950 - val_loss: 2.9424e-04 - val_accuracy: 0.9949 - lr: 6.2500e-05
Epoch 25/70
551/551 [==============================] - ETA: 0s - loss: 2.7524e-04 - accuracy:
0.9949
Epoch 25: val_loss improved from 0.00028 to 0.00027, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 79s 143ms/step - loss: 2.7524e-04 -
accuracy: 0.9949 - val_loss: 2.6615e-04 - val_accuracy: 0.9954 - lr: 6.2500e-05
Epoch 26/70
551/551 [==============================] - ETA: 0s - loss: 2.6221e-04 - accuracy:
0.9953
Epoch 26: val_loss did not improve from 0.00027
551/551 [==============================] - 79s 144ms/step - loss: 2.6221e-04 -
accuracy: 0.9953 - val_loss: 2.7259e-04 - val_accuracy: 0.9957 - lr: 6.2500e-05
Epoch 27/70
551/551 [==============================] - ETA: 0s - loss: 2.5933e-04 - accuracy:
0.9952
Epoch 27: val_loss improved from 0.00027 to 0.00026, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 146ms/step - loss: 2.5933e-04 -
accuracy: 0.9952 - val_loss: 2.6422e-04 - val_accuracy: 0.9957 - lr: 6.2500e-05
Epoch 28/70
551/551 [==============================] - ETA: 0s - loss: 2.5371e-04 - accuracy:
0.9953
Epoch 28: val_loss improved from 0.00026 to 0.00025, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 146ms/step - loss: 2.5371e-04 -
accuracy: 0.9953 - val_loss: 2.5252e-04 - val_accuracy: 0.9957 - lr: 6.2500e-05
Epoch 29/70
551/551 [==============================] - ETA: 0s - loss: 2.4545e-04 - accuracy:
0.9954
Epoch 29: val_loss improved from 0.00025 to 0.00025, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 145ms/step - loss: 2.4545e-04 -
accuracy: 0.9954 - val_loss: 2.5049e-04 - val_accuracy: 0.9960 - lr: 6.2500e-05
Epoch 30/70
551/551 [==============================] - ETA: 0s - loss: 2.4587e-04 - accuracy:
0.9955
Epoch 30: val_loss improved from 0.00025 to 0.00025, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 145ms/step - loss: 2.4587e-04 -
accuracy: 0.9955 - val_loss: 2.4838e-04 - val_accuracy: 0.9958 - lr: 6.2500e-05
Epoch 31/70
551/551 [==============================] - ETA: 0s - loss: 2.3814e-04 - accuracy:
0.9957
Epoch 31: val_loss improved from 0.00025 to 0.00025, saving model to ./CNN+LSTM.h5
```

```
551/551 [==============================] - 87s 158ms/step - loss: 2.3814e-04 -
accuracy: 0.9957 - val_loss: 2.4727e-04 - val_accuracy: 0.9959 - lr: 6.2500e-05
Epoch 32/70
551/551 [==============================] - ETA: 0s - loss: 2.3643e-04 - accuracy:
0.9957
Epoch 32: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.

Epoch 32: val_loss improved from 0.00025 to 0.00025, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 147ms/step - loss: 2.3643e-04 -
accuracy: 0.9957 - val_loss: 2.4605e-04 - val_accuracy: 0.9960 - lr: 6.2500e-05
Epoch 33/70
551/551 [==============================] - ETA: 0s - loss: 2.2498e-04 - accuracy:
0.9959
Epoch 33: val_loss improved from 0.00025 to 0.00024, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 145ms/step - loss: 2.2498e-04 -
accuracy: 0.9959 - val_loss: 2.3752e-04 - val_accuracy: 0.9958 - lr: 3.1250e-05
Epoch 34/70
551/551 [==============================] - ETA: 0s - loss: 2.2416e-04 - accuracy:
0.9959
Epoch 34: val_loss did not improve from 0.00024
551/551 [==============================] - 80s 145ms/step - loss: 2.2416e-04 -
accuracy: 0.9959 - val_loss: 2.4637e-04 - val_accuracy: 0.9958 - lr: 3.1250e-05
Epoch 35/70
551/551 [==============================] - ETA: 0s - loss: 2.1944e-04 - accuracy:
0.9961
Epoch 35: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.

Epoch 35: val_loss improved from 0.00024 to 0.00023, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 146ms/step - loss: 2.1944e-04 -
accuracy: 0.9961 - val_loss: 2.3445e-04 - val_accuracy: 0.9957 - lr: 3.1250e-05
Epoch 36/70
551/551 [==============================] - ETA: 0s - loss: 2.1608e-04 - accuracy:
0.9963
Epoch 36: val_loss improved from 0.00023 to 0.00023, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 145ms/step - loss: 2.1608e-04 -
accuracy: 0.9963 - val_loss: 2.3159e-04 - val_accuracy: 0.9964 - lr: 1.5625e-05
Epoch 37/70
551/551 [==============================] - ETA: 0s - loss: 2.1468e-04 - accuracy:
0.9963
Epoch 37: val_loss improved from 0.00023 to 0.00023, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 145ms/step - loss: 2.1468e-04 -
accuracy: 0.9963 - val_loss: 2.2733e-04 - val_accuracy: 0.9963 - lr: 1.5625e-05
Epoch 38/70
551/551 [==============================] - ETA: 0s - loss: 2.1210e-04 - accuracy:
0.9963
Epoch 38: val_loss did not improve from 0.00023
551/551 [==============================] - 80s 145ms/step - loss: 2.1210e-04 -
accuracy: 0.9963 - val_loss: 2.2817e-04 - val_accuracy: 0.9964 - lr: 1.5625e-05
Epoch 39/70
551/551 [==============================] - ETA: 0s - loss: 2.1626e-04 - accuracy:
0.9963
```

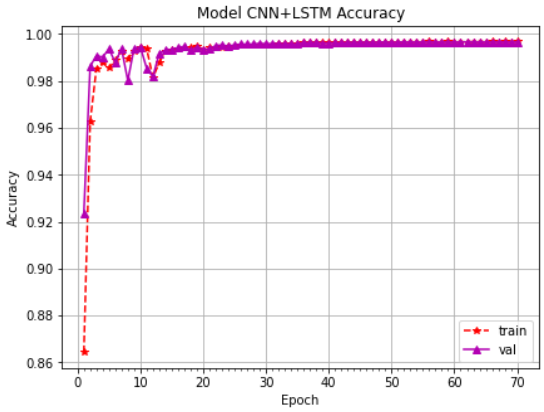
```
Epoch 39: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.

Epoch 39: val_loss did not improve from 0.00023
551/551 [==============================] - 80s 145ms/step - loss: 2.1626e-04 -
accuracy: 0.9963 - val_loss: 2.3199e-04 - val_accuracy: 0.9961 - lr: 1.5625e-05
Epoch 40/70
551/551 [==============================] - ETA: 0s - loss: 2.0932e-04 - accuracy:
0.9965
Epoch 40: val_loss did not improve from 0.00023
551/551 [==============================] - 80s 145ms/step - loss: 2.0932e-04 -
accuracy: 0.9965 - val_loss: 2.2974e-04 - val_accuracy: 0.9962 - lr: 7.8125e-06
Epoch 41/70
551/551 [==============================] - ETA: 0s - loss: 2.0864e-04 - accuracy:
0.9963
Epoch 41: val_loss improved from 0.00023 to 0.00023, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 148ms/step - loss: 2.0864e-04 -
accuracy: 0.9963 - val_loss: 2.2541e-04 - val_accuracy: 0.9963 - lr: 7.8125e-06
Epoch 42/70
551/551 [==============================] - ETA: 0s - loss: 2.0953e-04 - accuracy:
0.9965
Epoch 42: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.

Epoch 42: val_loss did not improve from 0.00023
551/551 [==============================] - 82s 148ms/step - loss: 2.0953e-04 -
accuracy: 0.9965 - val_loss: 2.2701e-04 - val_accuracy: 0.9962 - lr: 7.8125e-06
Epoch 43/70
551/551 [==============================] - ETA: 0s - loss: 2.0538e-04 - accuracy:
0.9966
Epoch 43: val_loss did not improve from 0.00023
551/551 [==============================] - 80s 146ms/step - loss: 2.0538e-04 -
accuracy: 0.9966 - val_loss: 2.2611e-04 - val_accuracy: 0.9964 - lr: 3.9063e-06
Epoch 44/70
551/551 [==============================] - ETA: 0s - loss: 2.0392e-04 - accuracy:
0.9966
Epoch 44: val_loss improved from 0.00023 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 146ms/step - loss: 2.0392e-04 -
accuracy: 0.9966 - val_loss: 2.2461e-04 - val_accuracy: 0.9964 - lr: 3.9063e-06
Epoch 45/70
551/551 [==============================] - ETA: 0s - loss: 2.0847e-04 - accuracy:
0.9965
Epoch 45: ReduceLROnPlateau reducing learning rate to 1.9531250927684596e-06.

Epoch 45: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 148ms/step - loss: 2.0847e-04 -
accuracy: 0.9965 - val_loss: 2.2453e-04 - val_accuracy: 0.9965 - lr: 3.9063e-06
Epoch 46/70
551/551 [==============================] - ETA: 0s - loss: 2.0648e-04 - accuracy:
0.9965
Epoch 46: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 82s 148ms/step - loss: 2.0648e-04 -
accuracy: 0.9965 - val_loss: 2.2339e-04 - val_accuracy: 0.9965 - lr: 1.9531e-06
```

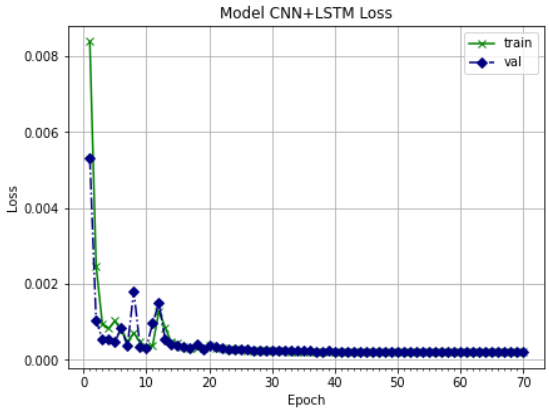
```
Epoch 47/70
551/551 [==============================] - ETA: 0s - loss: 2.0127e-04 - accuracy:
0.9966
Epoch 47: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 146ms/step - loss: 2.0127e-04 -
accuracy: 0.9966 - val_loss: 2.2355e-04 - val_accuracy: 0.9964 - lr: 1.9531e-06
Epoch 48/70
551/551 [==============================] - ETA: 0s - loss: 2.0425e-04 - accuracy:
0.9965
Epoch 48: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 82s 148ms/step - loss: 2.0425e-04 -
accuracy: 0.9965 - val_loss: 2.2284e-04 - val_accuracy: 0.9966 - lr: 1.9531e-06
Epoch 49/70
551/551 [==============================] - ETA: 0s - loss: 2.0501e-04 - accuracy:
0.9966
Epoch 49: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 82s 148ms/step - loss: 2.0501e-04 -
accuracy: 0.9966 - val_loss: 2.2255e-04 - val_accuracy: 0.9966 - lr: 1.9531e-06
Epoch 50/70
551/551 [==============================] - ETA: 0s - loss: 2.0323e-04 - accuracy:
0.9966
Epoch 50: val_loss did not improve from 0.00022
551/551 [==============================] - 82s 148ms/step - loss: 2.0323e-04 -
accuracy: 0.9966 - val_loss: 2.2480e-04 - val_accuracy: 0.9964 - lr: 1.9531e-06
Epoch 51/70
551/551 [==============================] - ETA: 0s - loss: 2.0172e-04 - accuracy:
0.9966
Epoch 51: ReduceLROnPlateau reducing learning rate to 9.765625463842298e-07.

Epoch 51: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 86s 156ms/step - loss: 2.0172e-04 -
accuracy: 0.9966 - val_loss: 2.2253e-04 - val_accuracy: 0.9966 - lr: 1.9531e-06
Epoch 52/70
551/551 [==============================] - ETA: 0s - loss: 2.0697e-04 - accuracy:
0.9967
Epoch 52: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 147ms/step - loss: 2.0697e-04 -
accuracy: 0.9967 - val_loss: 2.2234e-04 - val_accuracy: 0.9967 - lr: 9.7656e-07
Epoch 53/70
551/551 [==============================] - ETA: 0s - loss: 2.0304e-04 - accuracy:
0.9967
Epoch 53: val_loss did not improve from 0.00022
551/551 [==============================] - 80s 145ms/step - loss: 2.0304e-04 -
accuracy: 0.9967 - val_loss: 2.2243e-04 - val_accuracy: 0.9966 - lr: 9.7656e-07
Epoch 54/70
551/551 [==============================] - ETA: 0s - loss: 2.0144e-04 - accuracy:
0.9967
Epoch 54: ReduceLROnPlateau reducing learning rate to 4.882812731921149e-07.

Epoch 54: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 80s 145ms/step - loss: 2.0144e-04 -
```

```
accuracy: 0.9967 - val_loss: 2.2234e-04 - val_accuracy: 0.9966 - lr: 9.7656e-07
Epoch 55/70
551/551 [==============================] - ETA: 0s - loss: 2.0410e-04 - accuracy:
0.9966
Epoch 55: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 147ms/step - loss: 2.0410e-04 -
accuracy: 0.9966 - val_loss: 2.2232e-04 - val_accuracy: 0.9966 - lr: 4.8828e-07
Epoch 56/70
551/551 [==============================] - ETA: 0s - loss: 2.0040e-04 - accuracy:
0.9967
Epoch 56: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 148ms/step - loss: 2.0040e-04 -
accuracy: 0.9967 - val_loss: 2.2234e-04 - val_accuracy: 0.9967 - lr: 4.8828e-07
Epoch 57/70
551/551 [==============================] - ETA: 0s - loss: 2.0255e-04 - accuracy:
0.9967
Epoch 57: ReduceLROnPlateau reducing learning rate to 2.4414063659605745e-07.

Epoch 57: val_loss did not improve from 0.00022
551/551 [==============================] - 83s 150ms/step - loss: 2.0255e-04 -
accuracy: 0.9967 - val_loss: 2.2233e-04 - val_accuracy: 0.9966 - lr: 4.8828e-07
Epoch 58/70
551/551 [==============================] - ETA: 0s - loss: 2.0241e-04 - accuracy:
0.9966
Epoch 58: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 148ms/step - loss: 2.0241e-04 -
accuracy: 0.9966 - val_loss: 2.2224e-04 - val_accuracy: 0.9966 - lr: 2.4414e-07
Epoch 59/70
551/551 [==============================] - ETA: 0s - loss: 2.0259e-04 - accuracy:
0.9967
Epoch 59: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 147ms/step - loss: 2.0259e-04 -
accuracy: 0.9967 - val_loss: 2.2238e-04 - val_accuracy: 0.9966 - lr: 2.4414e-07
Epoch 60/70
551/551 [==============================] - ETA: 0s - loss: 2.0332e-04 - accuracy:
0.9967
Epoch 60: ReduceLROnPlateau reducing learning rate to 1.2207031829802872e-07.

Epoch 60: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 148ms/step - loss: 2.0332e-04 -
accuracy: 0.9967 - val_loss: 2.2214e-04 - val_accuracy: 0.9966 - lr: 2.4414e-07
Epoch 61/70
551/551 [==============================] - ETA: 0s - loss: 2.0040e-04 - accuracy:
0.9967
Epoch 61: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 148ms/step - loss: 2.0040e-04 -
accuracy: 0.9967 - val_loss: 2.2220e-04 - val_accuracy: 0.9966 - lr: 1.2207e-07
Epoch 62/70
551/551 [==============================] - ETA: 0s - loss: 2.0412e-04 - accuracy:
0.9967
Epoch 62: val_loss did not improve from 0.00022
```

```
551/551 [==============================] - 81s 147ms/step - loss: 2.0412e-04 -
accuracy: 0.9967 - val_loss: 2.2216e-04 - val_accuracy: 0.9967 - lr: 1.2207e-07
Epoch 63/70
551/551 [==============================] - ETA: 0s - loss: 2.0276e-04 - accuracy:
0.9967
Epoch 63: ReduceLROnPlateau reducing learning rate to 6.103515914901436e-08.

Epoch 63: val_loss did not improve from 0.00022
551/551 [==============================] - 83s 151ms/step - loss: 2.0276e-04 -
accuracy: 0.9967 - val_loss: 2.2216e-04 - val_accuracy: 0.9967 - lr: 1.2207e-07
Epoch 64/70
551/551 [==============================] - ETA: 0s - loss: 1.9999e-04 - accuracy:
0.9967
Epoch 64: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 83s 151ms/step - loss: 1.9999e-04 -
accuracy: 0.9967 - val_loss: 2.2213e-04 - val_accuracy: 0.9966 - lr: 6.1035e-08
Epoch 65/70
551/551 [==============================] - ETA: 0s - loss: 2.0532e-04 - accuracy:
0.9967
Epoch 65: val_loss did not improve from 0.00022
551/551 [==============================] - 82s 148ms/step - loss: 2.0532e-04 -
accuracy: 0.9967 - val_loss: 2.2215e-04 - val_accuracy: 0.9967 - lr: 6.1035e-08
Epoch 66/70
551/551 [==============================] - ETA: 0s - loss: 2.0121e-04 - accuracy:
0.9968
Epoch 66: ReduceLROnPlateau reducing learning rate to 3.051757957450718e-08.

Epoch 66: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 147ms/step - loss: 2.0121e-04 -
accuracy: 0.9968 - val_loss: 2.2213e-04 - val_accuracy: 0.9966 - lr: 6.1035e-08
Epoch 67/70
551/551 [==============================] - ETA: 0s - loss: 2.0104e-04 - accuracy:
0.9967
Epoch 67: val_loss improved from 0.00022 to 0.00022, saving model to ./CNN+LSTM.h5
551/551 [==============================] - 81s 147ms/step - loss: 2.0104e-04 -
accuracy: 0.9967 - val_loss: 2.2210e-04 - val_accuracy: 0.9966 - lr: 3.0518e-08
Epoch 68/70
551/551 [==============================] - ETA: 0s - loss: 2.0020e-04 - accuracy:
0.9967
Epoch 68: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 147ms/step - loss: 2.0020e-04 -
accuracy: 0.9967 - val_loss: 2.2211e-04 - val_accuracy: 0.9966 - lr: 3.0518e-08
Epoch 69/70
551/551 [==============================] - ETA: 0s - loss: 2.0349e-04 - accuracy:
0.9967
Epoch 69: ReduceLROnPlateau reducing learning rate to 1.525878978725359e-08.

Epoch 69: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 147ms/step - loss: 2.0349e-04 -
accuracy: 0.9967 - val_loss: 2.2212e-04 - val_accuracy: 0.9967 - lr: 3.0518e-08
Epoch 70/70
```

```
551/551 [==============================] - ETA: 0s - loss: 2.0142e-04 - accuracy:
0.9968
Epoch 70: val_loss did not improve from 0.00022
551/551 [==============================] - 81s 146ms/step - loss: 2.0142e-04 -
accuracy: 0.9968 - val_loss: 2.2213e-04 - val_accuracy: 0.9966 - lr: 1.5259e-08

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12:
MatplotlibDeprecationWarning: Passing the minor parameter of set_xticks()
positionally is deprecated since Matplotlib 3.2; the parameter will become
keyword-only two minor releases later.
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22:
MatplotlibDeprecationWarning: Passing the minor parameter of set_xticks()
positionally is deprecated since Matplotlib 3.2; the parameter will become
keyword-only two minor releases later.
```





 # Training Deep residual CNN model

model_val(model2, model2Name)

```
Epoch 1/70
551/551 [==============================] - ETA: 0s - loss: 0.0095 - accuracy:
0.8508
Epoch 1: val_loss improved from inf to 0.00536, saving model to ./Deep residual
CNN.h5
551/551 [==============================] - 127s 228ms/step - loss: 0.0095 -
accuracy: 0.8508 - val_loss: 0.0054 - val_accuracy: 0.9240 - lr: 0.0010
Epoch 2/70
551/551 [==============================] - ETA: 0s - loss: 0.0024 - accuracy:
0.9662
Epoch 2: val_loss improved from 0.00536 to 0.00082, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 126s 228ms/step - loss: 0.0024 -
accuracy: 0.9662 - val_loss: 8.2114e-04 - val_accuracy: 0.9884 - lr: 0.0010
Epoch 3/70
551/551 [==============================] - ETA: 0s - loss: 0.0013 - accuracy:
0.9817
Epoch 3: val_loss improved from 0.00082 to 0.00071, saving model to ./Deep
residual CNN.h5
```

```
551/551 [==============================] - 125s 227ms/step - loss: 0.0013 -
accuracy: 0.9817 - val_loss: 7.1186e-04 - val_accuracy: 0.9904 - lr: 0.0010
Epoch 4/70
551/551 [==============================] - ETA: 0s - loss: 0.0013 - accuracy:
0.9834
Epoch 4: val_loss did not improve from 0.00071
551/551 [==============================] - 124s 225ms/step - loss: 0.0013 -
accuracy: 0.9834 - val_loss: 9.0621e-04 - val_accuracy: 0.9851 - lr: 0.0010
Epoch 5/70
551/551 [==============================] - ETA: 0s - loss: 9.7973e-04 - accuracy:
0.9868
Epoch 5: val_loss improved from 0.00071 to 0.00056, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 124s 224ms/step - loss: 9.7973e-04 -
accuracy: 0.9868 - val_loss: 5.6065e-04 - val_accuracy: 0.9921 - lr: 0.0010
Epoch 6/70
551/551 [==============================] - ETA: 0s - loss: 9.1613e-04 - accuracy:
0.9870
Epoch 6: val_loss did not improve from 0.00056
551/551 [==============================] - 124s 225ms/step - loss: 9.1613e-04 -
accuracy: 0.9870 - val_loss: 7.1180e-04 - val_accuracy: 0.9881 - lr: 0.0010
Epoch 7/70
551/551 [==============================] - ETA: 0s - loss: 8.9702e-04 - accuracy:
0.9878
Epoch 7: val_loss did not improve from 0.00056
551/551 [==============================] - 124s 225ms/step - loss: 8.9702e-04 -
accuracy: 0.9878 - val_loss: 8.1508e-04 - val_accuracy: 0.9899 - lr: 0.0010
Epoch 8/70
551/551 [==============================] - ETA: 0s - loss: 6.3135e-04 - accuracy:
0.9911
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 8: val_loss did not improve from 0.00056
551/551 [==============================] - 123s 224ms/step - loss: 6.3135e-04 -
accuracy: 0.9911 - val_loss: 0.0043 - val_accuracy: 0.9422 - lr: 0.0010
Epoch 9/70
551/551 [==============================] - ETA: 0s - loss: 5.3564e-04 - accuracy:
0.9918
Epoch 9: val_loss improved from 0.00056 to 0.00032, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 124s 225ms/step - loss: 5.3564e-04 -
accuracy: 0.9918 - val_loss: 3.1736e-04 - val_accuracy: 0.9948 - lr: 5.0000e-04
Epoch 10/70
551/551 [==============================] - ETA: 0s - loss: 3.3955e-04 - accuracy:
0.9940
Epoch 10: val_loss did not improve from 0.00032
551/551 [==============================] - 124s 225ms/step - loss: 3.3955e-04 -
accuracy: 0.9940 - val_loss: 3.5456e-04 - val_accuracy: 0.9938 - lr: 5.0000e-04
Epoch 11/70
551/551 [==============================] - ETA: 0s - loss: 3.9738e-04 - accuracy:
0.9935
```

```
Epoch 11: val_loss did not improve from 0.00032
551/551 [==============================] - 124s 224ms/step - loss: 3.9738e-04 -
accuracy: 0.9935 - val_loss: 3.4224e-04 - val_accuracy: 0.9932 - lr: 5.0000e-04
Epoch 12/70
551/551 [==============================] - ETA: 0s - loss: 4.3741e-04 - accuracy:
0.9930
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 12: val_loss did not improve from 0.00032
551/551 [==============================] - 123s 224ms/step - loss: 4.3741e-04 -
accuracy: 0.9930 - val_loss: 3.5104e-04 - val_accuracy: 0.9941 - lr: 5.0000e-04
Epoch 13/70
551/551 [==============================] - ETA: 0s - loss: 2.6570e-04 - accuracy:
0.9952
Epoch 13: val_loss improved from 0.00032 to 0.00026, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 121s 219ms/step - loss: 2.6570e-04 -
accuracy: 0.9952 - val_loss: 2.5947e-04 - val_accuracy: 0.9957 - lr: 2.5000e-04
Epoch 14/70
551/551 [==============================] - ETA: 0s - loss: 2.6826e-04 - accuracy:
0.9951
Epoch 14: val_loss improved from 0.00026 to 0.00025, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 118s 214ms/step - loss: 2.6826e-04 -
accuracy: 0.9951 - val_loss: 2.4988e-04 - val_accuracy: 0.9953 - lr: 2.5000e-04
Epoch 15/70
551/551 [==============================] - ETA: 0s - loss: 2.6474e-04 - accuracy:
0.9949
Epoch 15: val_loss did not improve from 0.00025
551/551 [==============================] - 121s 219ms/step - loss: 2.6474e-04 -
accuracy: 0.9949 - val_loss: 2.8179e-04 - val_accuracy: 0.9953 - lr: 2.5000e-04
Epoch 16/70
551/551 [==============================] - ETA: 0s - loss: 2.6519e-04 - accuracy:
0.9950
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.

Epoch 16: val_loss did not improve from 0.00025
551/551 [==============================] - 119s 216ms/step - loss: 2.6519e-04 -
accuracy: 0.9950 - val_loss: 3.4670e-04 - val_accuracy: 0.9918 - lr: 2.5000e-04
Epoch 17/70
551/551 [==============================] - ETA: 0s - loss: 2.2247e-04 - accuracy:
0.9959
Epoch 17: val_loss did not improve from 0.00025
551/551 [==============================] - 120s 217ms/step - loss: 2.2247e-04 -
accuracy: 0.9959 - val_loss: 2.5491e-04 - val_accuracy: 0.9949 - lr: 1.2500e-04
Epoch 18/70
551/551 [==============================] - ETA: 0s - loss: 2.3949e-04 - accuracy:
0.9956
Epoch 18: val_loss improved from 0.00025 to 0.00024, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 118s 215ms/step - loss: 2.3949e-04 -
```

```
accuracy: 0.9956 - val_loss: 2.3571e-04 - val_accuracy: 0.9957 - lr: 1.2500e-04
Epoch 19/70
551/551 [==============================] - ETA: 0s - loss: 2.1961e-04 - accuracy:
0.9959
Epoch 19: val_loss improved from 0.00024 to 0.00023, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 121s 220ms/step - loss: 2.1961e-04 -
accuracy: 0.9959 - val_loss: 2.3181e-04 - val_accuracy: 0.9959 - lr: 1.2500e-04
Epoch 20/70
551/551 [==============================] - ETA: 0s - loss: 2.1851e-04 - accuracy:
0.9958
Epoch 20: val_loss improved from 0.00023 to 0.00022, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 121s 219ms/step - loss: 2.1851e-04 -
accuracy: 0.9958 - val_loss: 2.1792e-04 - val_accuracy: 0.9962 - lr: 1.2500e-04
Epoch 21/70
551/551 [==============================] - ETA: 0s - loss: 2.1025e-04 - accuracy:
0.9960
Epoch 21: val_loss did not improve from 0.00022
551/551 [==============================] - 119s 216ms/step - loss: 2.1025e-04 -
accuracy: 0.9960 - val_loss: 2.2145e-04 - val_accuracy: 0.9960 - lr: 1.2500e-04
Epoch 22/70
551/551 [==============================] - ETA: 0s - loss: 2.1704e-04 - accuracy:
0.9957
Epoch 22: val_loss did not improve from 0.00022
551/551 [==============================] - 118s 215ms/step - loss: 2.1704e-04 -
accuracy: 0.9957 - val_loss: 2.6106e-04 - val_accuracy: 0.9956 - lr: 1.2500e-04
Epoch 23/70
551/551 [==============================] - ETA: 0s - loss: 2.1953e-04 - accuracy:
0.9957
Epoch 23: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

Epoch 23: val_loss did not improve from 0.00022
551/551 [==============================] - 118s 214ms/step - loss: 2.1953e-04 -
accuracy: 0.9957 - val_loss: 2.2598e-04 - val_accuracy: 0.9961 - lr: 1.2500e-04
Epoch 24/70
551/551 [==============================] - ETA: 0s - loss: 1.8426e-04 - accuracy:
0.9966
Epoch 24: val_loss did not improve from 0.00022
551/551 [==============================] - 118s 213ms/step - loss: 1.8426e-04 -
accuracy: 0.9966 - val_loss: 2.3709e-04 - val_accuracy: 0.9957 - lr: 6.2500e-05
Epoch 25/70
551/551 [==============================] - ETA: 0s - loss: 1.7958e-04 - accuracy:
0.9967
Epoch 25: val_loss improved from 0.00022 to 0.00020, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 118s 214ms/step - loss: 1.7958e-04 -
accuracy: 0.9967 - val_loss: 2.0021e-04 - val_accuracy: 0.9967 - lr: 6.2500e-05
Epoch 26/70
551/551 [==============================] - ETA: 0s - loss: 1.7866e-04 - accuracy:
0.9966
```

```
Epoch 26: val_loss improved from 0.00020 to 0.00020, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 117s 213ms/step - loss: 1.7866e-04 -
accuracy: 0.9966 - val_loss: 1.9865e-04 - val_accuracy: 0.9965 - lr: 6.2500e-05
Epoch 27/70
551/551 [==============================] - ETA: 0s - loss: 1.7890e-04 - accuracy:
0.9967
Epoch 27: val_loss did not improve from 0.00020
551/551 [==============================] - 117s 213ms/step - loss: 1.7890e-04 -
accuracy: 0.9967 - val_loss: 2.2419e-04 - val_accuracy: 0.9954 - lr: 6.2500e-05
Epoch 28/70
551/551 [==============================] - ETA: 0s - loss: 1.8010e-04 - accuracy:
0.9966
Epoch 28: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.

Epoch 28: val_loss did not improve from 0.00020
551/551 [==============================] - 117s 213ms/step - loss: 1.8010e-04 -
accuracy: 0.9966 - val_loss: 2.0181e-04 - val_accuracy: 0.9965 - lr: 6.2500e-05
Epoch 29/70
551/551 [==============================] - ETA: 0s - loss: 1.5448e-04 - accuracy:
0.9973
Epoch 29: val_loss improved from 0.00020 to 0.00020, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 118s 214ms/step - loss: 1.5448e-04 -
accuracy: 0.9973 - val_loss: 1.9858e-04 - val_accuracy: 0.9964 - lr: 3.1250e-05
Epoch 30/70
551/551 [==============================] - ETA: 0s - loss: 1.5841e-04 - accuracy:
0.9970
Epoch 30: val_loss improved from 0.00020 to 0.00018, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 117s 212ms/step - loss: 1.5841e-04 -
accuracy: 0.9970 - val_loss: 1.8227e-04 - val_accuracy: 0.9967 - lr: 3.1250e-05
Epoch 31/70
551/551 [==============================] - ETA: 0s - loss: 1.5294e-04 - accuracy:
0.9972
Epoch 31: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.

Epoch 31: val_loss did not improve from 0.00018
551/551 [==============================] - 118s 214ms/step - loss: 1.5294e-04 -
accuracy: 0.9972 - val_loss: 1.8491e-04 - val_accuracy: 0.9968 - lr: 3.1250e-05
Epoch 32/70
551/551 [==============================] - ETA: 0s - loss: 1.4168e-04 - accuracy:
0.9975
Epoch 32: val_loss improved from 0.00018 to 0.00017, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 118s 214ms/step - loss: 1.4168e-04 -
accuracy: 0.9975 - val_loss: 1.6830e-04 - val_accuracy: 0.9971 - lr: 1.5625e-05
Epoch 33/70
551/551 [==============================] - ETA: 0s - loss: 1.3969e-04 - accuracy:
0.9976
Epoch 33: val_loss did not improve from 0.00017
```

```
551/551 [==============================] - 118s 215ms/step - loss: 1.3969e-04 -
accuracy: 0.9976 - val_loss: 1.8636e-04 - val_accuracy: 0.9968 - lr: 1.5625e-05
Epoch 34/70
551/551 [==============================] - ETA: 0s - loss: 1.3972e-04 - accuracy:
0.9975
Epoch 34: val_loss did not improve from 0.00017
551/551 [==============================] - 119s 216ms/step - loss: 1.3972e-04 -
accuracy: 0.9975 - val_loss: 1.7667e-04 - val_accuracy: 0.9970 - lr: 1.5625e-05
Epoch 35/70
551/551 [==============================] - ETA: 0s - loss: 1.3753e-04 - accuracy:
0.9975
Epoch 35: val_loss did not improve from 0.00017
551/551 [==============================] - 119s 215ms/step - loss: 1.3753e-04 -
accuracy: 0.9975 - val_loss: 1.8109e-04 - val_accuracy: 0.9976 - lr: 1.5625e-05
Epoch 36/70
551/551 [==============================] - ETA: 0s - loss: 1.3563e-04 - accuracy:
0.9977
Epoch 36: val_loss did not improve from 0.00017
551/551 [==============================] - 119s 216ms/step - loss: 1.3563e-04 -
accuracy: 0.9977 - val_loss: 1.6924e-04 - val_accuracy: 0.9973 - lr: 1.5625e-05
Epoch 37/70
551/551 [==============================] - ETA: 0s - loss: 1.3339e-04 - accuracy:
0.9977
Epoch 37: val_loss did not improve from 0.00017
551/551 [==============================] - 119s 216ms/step - loss: 1.3339e-04 -
accuracy: 0.9977 - val_loss: 1.7892e-04 - val_accuracy: 0.9970 - lr: 1.5625e-05
Epoch 38/70
551/551 [==============================] - ETA: 0s - loss: 1.3506e-04 - accuracy:
0.9977
Epoch 38: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.

Epoch 38: val_loss improved from 0.00017 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 119s 217ms/step - loss: 1.3506e-04 -
accuracy: 0.9977 - val_loss: 1.6257e-04 - val_accuracy: 0.9975 - lr: 1.5625e-05
Epoch 39/70
551/551 [==============================] - ETA: 0s - loss: 1.2474e-04 - accuracy:
0.9980
Epoch 39: val_loss did not improve from 0.00016
551/551 [==============================] - 120s 218ms/step - loss: 1.2474e-04 -
accuracy: 0.9980 - val_loss: 1.6401e-04 - val_accuracy: 0.9973 - lr: 7.8125e-06
Epoch 40/70
551/551 [==============================] - ETA: 0s - loss: 1.2679e-04 - accuracy:
0.9978
Epoch 40: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 120s 218ms/step - loss: 1.2679e-04 -
accuracy: 0.9978 - val_loss: 1.6232e-04 - val_accuracy: 0.9974 - lr: 7.8125e-06
Epoch 41/70
551/551 [==============================] - ETA: 0s - loss: 1.2326e-04 - accuracy:
0.9980
```

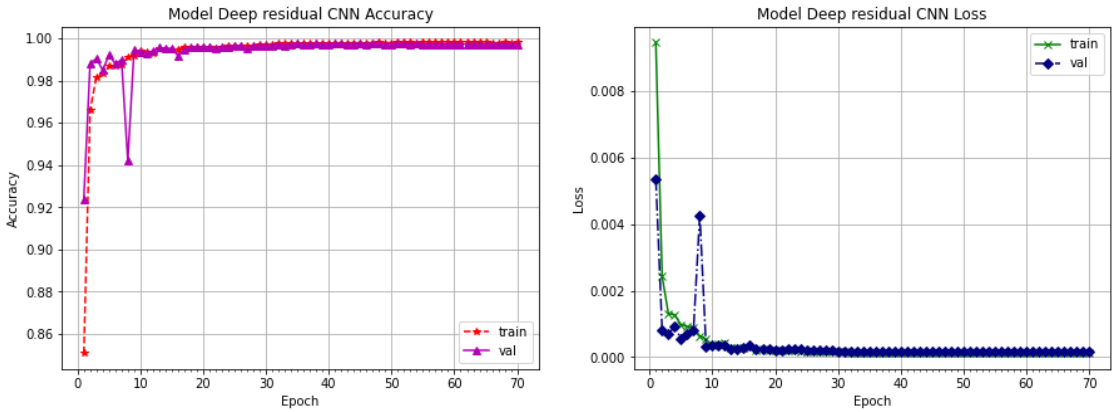
```
Epoch 41: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.

Epoch 41: val_loss did not improve from 0.00016
551/551 [==============================] - 121s 220ms/step - loss: 1.2326e-04 -
accuracy: 0.9980 - val_loss: 1.6693e-04 - val_accuracy: 0.9975 - lr: 7.8125e-06
Epoch 42/70
551/551 [==============================] - ETA: 0s - loss: 1.2178e-04 - accuracy:
0.9980
Epoch 42: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 123s 222ms/step - loss: 1.2178e-04 -
accuracy: 0.9980 - val_loss: 1.6046e-04 - val_accuracy: 0.9975 - lr: 3.9063e-06
Epoch 43/70
551/551 [==============================] - ETA: 0s - loss: 1.2215e-04 - accuracy:
0.9980
Epoch 43: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 124s 224ms/step - loss: 1.2215e-04 -
accuracy: 0.9980 - val_loss: 1.5964e-04 - val_accuracy: 0.9973 - lr: 3.9063e-06
Epoch 44/70
551/551 [==============================] - ETA: 0s - loss: 1.1900e-04 - accuracy:
0.9980
Epoch 44: ReduceLROnPlateau reducing learning rate to 1.9531250927684596e-06.

Epoch 44: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 122s 222ms/step - loss: 1.1900e-04 -
accuracy: 0.9980 - val_loss: 1.5926e-04 - val_accuracy: 0.9974 - lr: 3.9063e-06
Epoch 45/70
551/551 [==============================] - ETA: 0s - loss: 1.1951e-04 - accuracy:
0.9980
Epoch 45: val_loss did not improve from 0.00016
551/551 [==============================] - 122s 222ms/step - loss: 1.1951e-04 -
accuracy: 0.9980 - val_loss: 1.6011e-04 - val_accuracy: 0.9976 - lr: 1.9531e-06
Epoch 46/70
551/551 [==============================] - ETA: 0s - loss: 1.1911e-04 - accuracy:
0.9980
Epoch 46: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 123s 224ms/step - loss: 1.1911e-04 -
accuracy: 0.9980 - val_loss: 1.5814e-04 - val_accuracy: 0.9974 - lr: 1.9531e-06
Epoch 47/70
551/551 [==============================] - ETA: 0s - loss: 1.1763e-04 - accuracy:
0.9980
Epoch 47: ReduceLROnPlateau reducing learning rate to 9.765625463842298e-07.

Epoch 47: val_loss did not improve from 0.00016
551/551 [==============================] - 123s 224ms/step - loss: 1.1763e-04 -
accuracy: 0.9980 - val_loss: 1.6033e-04 - val_accuracy: 0.9975 - lr: 1.9531e-06
Epoch 48/70
551/551 [==============================] - ETA: 0s - loss: 1.1608e-04 - accuracy:
```

```
0.9981
Epoch 48: val_loss did not improve from 0.00016
551/551 [==============================] - 124s 224ms/step - loss: 1.1608e-04 -
accuracy: 0.9981 - val_loss: 1.5984e-04 - val_accuracy: 0.9975 - lr: 9.7656e-07
Epoch 49/70
551/551 [==============================] - ETA: 0s - loss: 1.1704e-04 - accuracy:
0.9980
Epoch 49: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 124s 225ms/step - loss: 1.1704e-04 -
accuracy: 0.9980 - val_loss: 1.5781e-04 - val_accuracy: 0.9974 - lr: 9.7656e-07
Epoch 50/70
551/551 [==============================] - ETA: 0s - loss: 1.1740e-04 - accuracy:
0.9980
Epoch 50: ReduceLROnPlateau reducing learning rate to 4.882812731921149e-07.

Epoch 50: val_loss did not improve from 0.00016
551/551 [==============================] - 123s 222ms/step - loss: 1.1740e-04 -
accuracy: 0.9980 - val_loss: 1.5802e-04 - val_accuracy: 0.9974 - lr: 9.7656e-07
Epoch 51/70
551/551 [==============================] - ETA: 0s - loss: 1.1744e-04 - accuracy:
0.9981
Epoch 51: val_loss did not improve from 0.00016
551/551 [==============================] - 123s 223ms/step - loss: 1.1744e-04 -
accuracy: 0.9981 - val_loss: 1.5805e-04 - val_accuracy: 0.9975 - lr: 4.8828e-07
Epoch 52/70
551/551 [==============================] - ETA: 0s - loss: 1.1615e-04 - accuracy:
0.9981
Epoch 52: val_loss did not improve from 0.00016
551/551 [==============================] - 122s 221ms/step - loss: 1.1615e-04 -
accuracy: 0.9981 - val_loss: 1.5787e-04 - val_accuracy: 0.9975 - lr: 4.8828e-07
Epoch 53/70
551/551 [==============================] - ETA: 0s - loss: 1.1577e-04 - accuracy:
0.9981
Epoch 53: ReduceLROnPlateau reducing learning rate to 2.4414063659605745e-07.

Epoch 53: val_loss did not improve from 0.00016
551/551 [==============================] - 122s 222ms/step - loss: 1.1577e-04 -
accuracy: 0.9981 - val_loss: 1.5788e-04 - val_accuracy: 0.9974 - lr: 4.8828e-07
Epoch 54/70
551/551 [==============================] - ETA: 0s - loss: 1.1853e-04 - accuracy:
0.9980
Epoch 54: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 121s 220ms/step - loss: 1.1853e-04 -
accuracy: 0.9980 - val_loss: 1.5774e-04 - val_accuracy: 0.9975 - lr: 2.4414e-07
Epoch 55/70
551/551 [==============================] - ETA: 0s - loss: 1.1507e-04 - accuracy:
0.9981
Epoch 55: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
```

```
551/551 [==============================] - 121s 220ms/step - loss: 1.1507e-04 -
accuracy: 0.9981 - val_loss: 1.5761e-04 - val_accuracy: 0.9974 - lr: 2.4414e-07
Epoch 56/70
551/551 [==============================] - ETA: 0s - loss: 1.1500e-04 - accuracy:
0.9981
Epoch 56: ReduceLROnPlateau reducing learning rate to 1.2207031829802872e-07.

Epoch 56: val_loss improved from 0.00016 to 0.00016, saving model to ./Deep
residual CNN.h5
551/551 [==============================] - 122s 221ms/step - loss: 1.1500e-04 -
accuracy: 0.9981 - val_loss: 1.5758e-04 - val_accuracy: 0.9974 - lr: 2.4414e-07
Epoch 57/70
551/551 [==============================] - ETA: 0s - loss: 1.1798e-04 - accuracy:
0.9981
Epoch 57: val_loss did not improve from 0.00016
551/551 [==============================] - 121s 219ms/step - loss: 1.1798e-04 -
accuracy: 0.9981 - val_loss: 1.5769e-04 - val_accuracy: 0.9974 - lr: 1.2207e-07
Epoch 58/70
551/551 [==============================] - ETA: 0s - loss: 1.1428e-04 - accuracy:
0.9981
Epoch 58: val_loss did not improve from 0.00016
551/551 [==============================] - 120s 217ms/step - loss: 1.1428e-04 -
accuracy: 0.9981 - val_loss: 1.5785e-04 - val_accuracy: 0.9974 - lr: 1.2207e-07
Epoch 59/70
551/551 [==============================] - ETA: 0s - loss: 1.1661e-04 - accuracy:
0.9981
Epoch 59: ReduceLROnPlateau reducing learning rate to 6.103515914901436e-08.

Epoch 59: val_loss did not improve from 0.00016
551/551 [==============================] - 120s 218ms/step - loss: 1.1661e-04 -
accuracy: 0.9981 - val_loss: 1.5781e-04 - val_accuracy: 0.9974 - lr: 1.2207e-07
Epoch 60/70
551/551 [==============================] - ETA: 0s - loss: 1.1744e-04 - accuracy:
0.9981
Epoch 60: val_loss did not improve from 0.00016
551/551 [==============================] - 122s 221ms/step - loss: 1.1744e-04 -
accuracy: 0.9981 - val_loss: 1.5771e-04 - val_accuracy: 0.9974 - lr: 6.1035e-08
Epoch 61/70
551/551 [==============================] - ETA: 0s - loss: 1.1554e-04 - accuracy:
0.9981
Epoch 61: val_loss did not improve from 0.00016
551/551 [==============================] - 121s 219ms/step - loss: 1.1554e-04 -
accuracy: 0.9981 - val_loss: 1.5769e-04 - val_accuracy: 0.9974 - lr: 6.1035e-08
Epoch 62/70
551/551 [==============================] - ETA: 0s - loss: 1.1577e-04 - accuracy:
0.9981
Epoch 62: ReduceLROnPlateau reducing learning rate to 3.051757957450718e-08.

Epoch 62: val_loss did not improve from 0.00016
551/551 [==============================] - 122s 221ms/step - loss: 1.1577e-04 -
accuracy: 0.9981 - val_loss: 1.5767e-04 - val_accuracy: 0.9974 - lr: 6.1035e-08
```

```
Epoch 63/70
551/551 [==============================] - ETA: 0s - loss: 1.1570e-04 - accuracy:
0.9980
Epoch 63: val_loss did not improve from 0.00016
551/551 [==============================] - 121s 220ms/step - loss: 1.1570e-04 -
accuracy: 0.9980 - val_loss: 1.5768e-04 - val_accuracy: 0.9974 - lr: 3.0518e-08
Epoch 64/70
551/551 [==============================] - ETA: 0s - loss: 1.1559e-04 - accuracy:
0.9981
Epoch 64: val_loss did not improve from 0.00016
551/551 [==============================] - 121s 220ms/step - loss: 1.1559e-04 -
accuracy: 0.9981 - val_loss: 1.5768e-04 - val_accuracy: 0.9974 - lr: 3.0518e-08
Epoch 65/70
551/551 [==============================] - ETA: 0s - loss: 1.1579e-04 - accuracy:
0.9981
Epoch 65: ReduceLROnPlateau reducing learning rate to 1.525878978725359e-08.

Epoch 65: val_loss did not improve from 0.00016
551/551 [==============================] - 124s 225ms/step - loss: 1.1579e-04 -
accuracy: 0.9981 - val_loss: 1.5762e-04 - val_accuracy: 0.9974 - lr: 3.0518e-08
Epoch 66/70
551/551 [==============================] - ETA: 0s - loss: 1.1628e-04 - accuracy:
0.9980
Epoch 66: val_loss did not improve from 0.00016
551/551 [==============================] - 122s 222ms/step - loss: 1.1628e-04 -
accuracy: 0.9980 - val_loss: 1.5763e-04 - val_accuracy: 0.9974 - lr: 1.5259e-08
Epoch 67/70
551/551 [==============================] - ETA: 0s - loss: 1.1697e-04 - accuracy:
0.9980
Epoch 67: val_loss did not improve from 0.00016
551/551 [==============================] - 121s 220ms/step - loss: 1.1697e-04 -
accuracy: 0.9980 - val_loss: 1.5765e-04 - val_accuracy: 0.9974 - lr: 1.5259e-08
Epoch 68/70
551/551 [==============================] - ETA: 0s - loss: 1.1348e-04 - accuracy:
0.9981
Epoch 68: ReduceLROnPlateau reducing learning rate to 7.629394893626795e-09.

Epoch 68: val_loss did not improve from 0.00016
551/551 [==============================] - 123s 223ms/step - loss: 1.1348e-04 -
accuracy: 0.9981 - val_loss: 1.5765e-04 - val_accuracy: 0.9974 - lr: 1.5259e-08
Epoch 69/70
551/551 [==============================] - ETA: 0s - loss: 1.1865e-04 - accuracy:
0.9980
Epoch 69: val_loss did not improve from 0.00016
551/551 [==============================] - 120s 218ms/step - loss: 1.1865e-04 -
accuracy: 0.9980 - val_loss: 1.5764e-04 - val_accuracy: 0.9974 - lr: 7.6294e-09
Epoch 70/70
551/551 [==============================] - ETA: 0s - loss: 1.1457e-04 - accuracy:
0.9981
Epoch 70: val_loss did not improve from 0.00016
```

```
551/551 [==============================] - 123s 224ms/step - loss: 1.1457e-04 -
accuracy: 0.9981 - val_loss: 1.5764e-04 - val_accuracy: 0.9974 - lr: 7.6294e-09

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12:
MatplotlibDeprecationWarning: Passing the minor parameter of set_xticks()
positionally is deprecated since Matplotlib 3.2; the parameter will become
keyword-only two minor releases later.
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22:
MatplotlibDeprecationWarning: Passing the minor parameter of set_xticks()
positionally is deprecated since Matplotlib 3.2; the parameter will become
keyword-only two minor releases later.
```



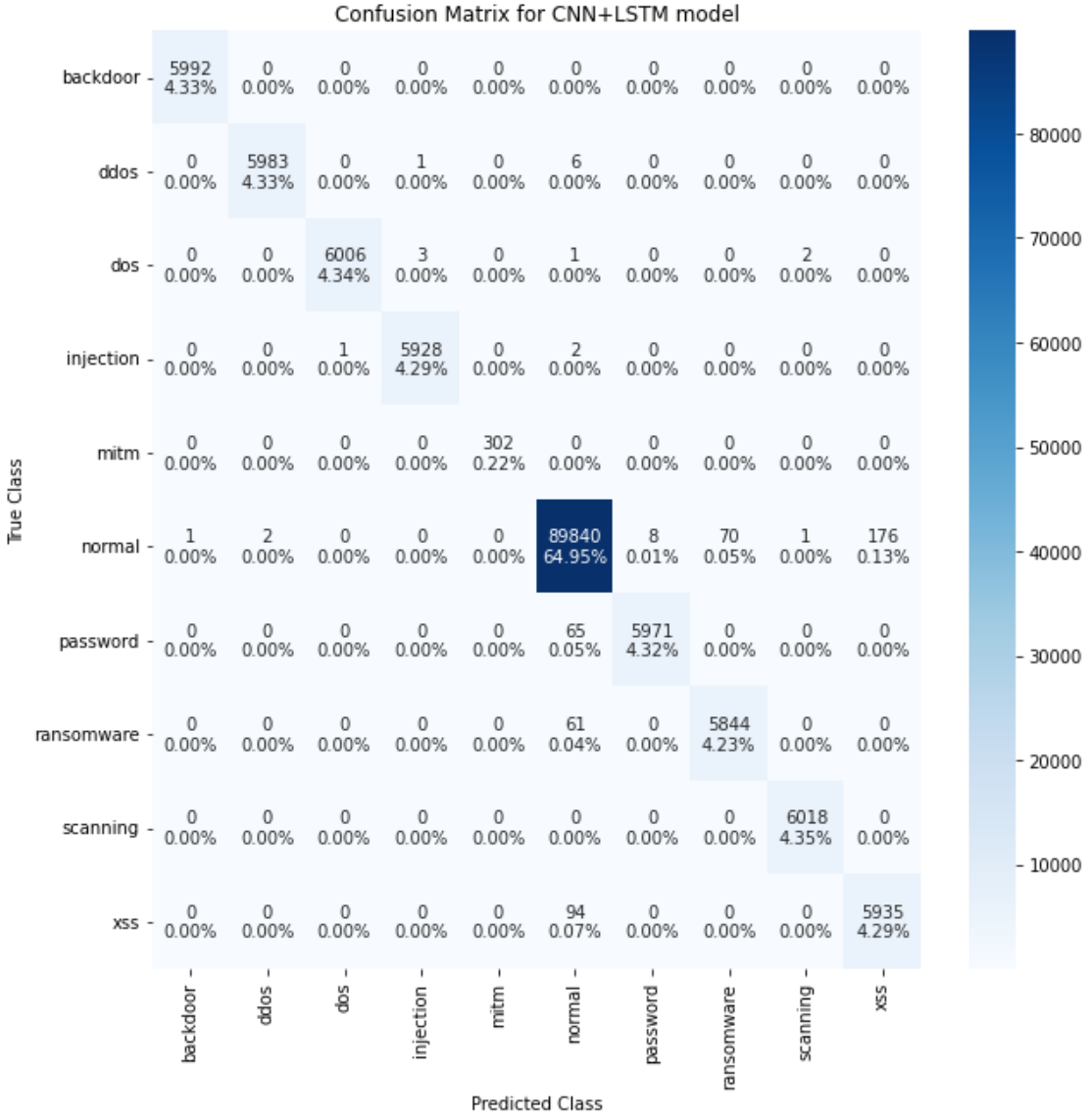
Testing the Models

```
# Function to test the models and calculate the Evaluation metrics


def eval_metrics(model, modelName):

    y_pred = model.predict(x_test)

    y_pred_cm  = np.argmax(y_pred, axis=1)
    y_test_cm  = np.argmax(y_test, axis=1)

    # Confusion matrix for the model
    cm = confusion_matrix(y_test_cm, y_pred_cm)

    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]

    group_percentages = ["{0:.2%}".format(value) for value in
cm.flatten()/np.sum(cm)]

    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts,group_percentages)]

    labels = np.asarray(labels).reshape(10,10)
    #labels = np.asarray(labels).reshape(2,2)
    label = ['backdoor', 'ddos', 'dos', 'injection', 'mitm', 'normal', 'password',
'ransomware', 'scanning', 'xss']
```

```
    #label = ['normal','malware']
    plt.figure(figsize=(10,10))
    #plt.figure(figsize=(2,2))
    sns.heatmap(cm, xticklabels=label, yticklabels=label, annot=labels, fmt='',
cmap="Blues", vmin = 0.2);
    plt.title('Confusion Matrix for '+ modelName+' model')
    plt.ylabel('True Class')
    plt.xlabel('Predicted Class')
    plt.savefig('./'+modelName+'_CM.png')
    plt.show()

    # Creating Classification Report with Accuracy and Precision of the model
    print(classification_report(y_test_cm, y_pred_cm, target_names= ['backdoor',
'ddos', 'dos', 'injection', 'mitm', 'normal', 'password', 'ransomware',
'scanning', 'xss'], digits=4))
    #print(classification_report(y_test_cm, y_pred_cm, target_names=
['normal','malware']))
    loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
    print("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))
    with open('./'+modelName+'_CR.txt','a') as f:
      f.write(classification_report(y_test_cm, y_pred_cm, target_names=
['backdoor', 'ddos', 'dos', 'injection', 'mitm', 'normal', 'password',
'ransomware', 'scanning', 'xss']))
      #f.write(classification_report(y_test_cm, y_pred_cm, target_names=
['normal','malware']))
      f.write("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))

    # Creating ROC curve for the model
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(labels.shape[1]):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
    colors = cycle(['blue', 'red', 'green','aqua', 'darkorange',
'orange','fuchsia', 'lime','magenta'])
    for i, color in zip(range(labels.shape[1]), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label='ROC curve of class {0} (area = {1:0.2f})'
                 ''.format(i, roc_auc[i]))
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.ylabel('Recall')
    plt.xlabel('Fall-out (1-Specificity)')
    plt.title('Receiver Operating Characteristic (ROC) for '+modelName+' model')
    plt.legend(loc="lower right")
    plt.savefig('./'+modelName+'_ROC.png')

    plt.show()
```

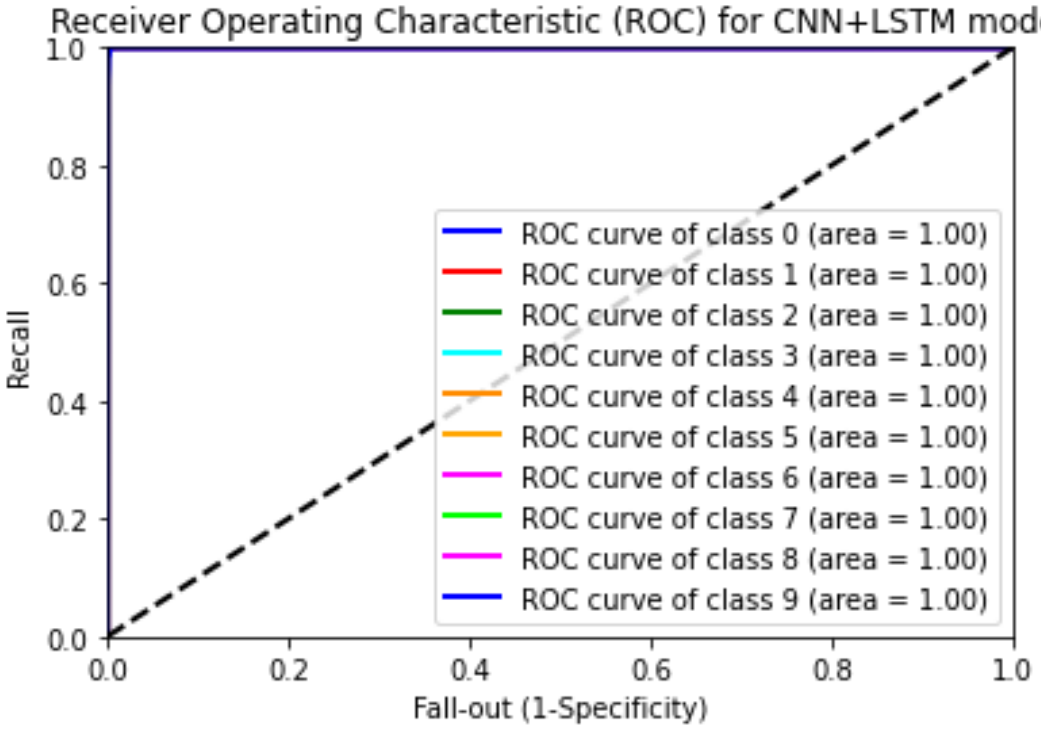
 # Evaluation Metrics of CNN-LSTM model

eval_metrics(model1, model1Name)



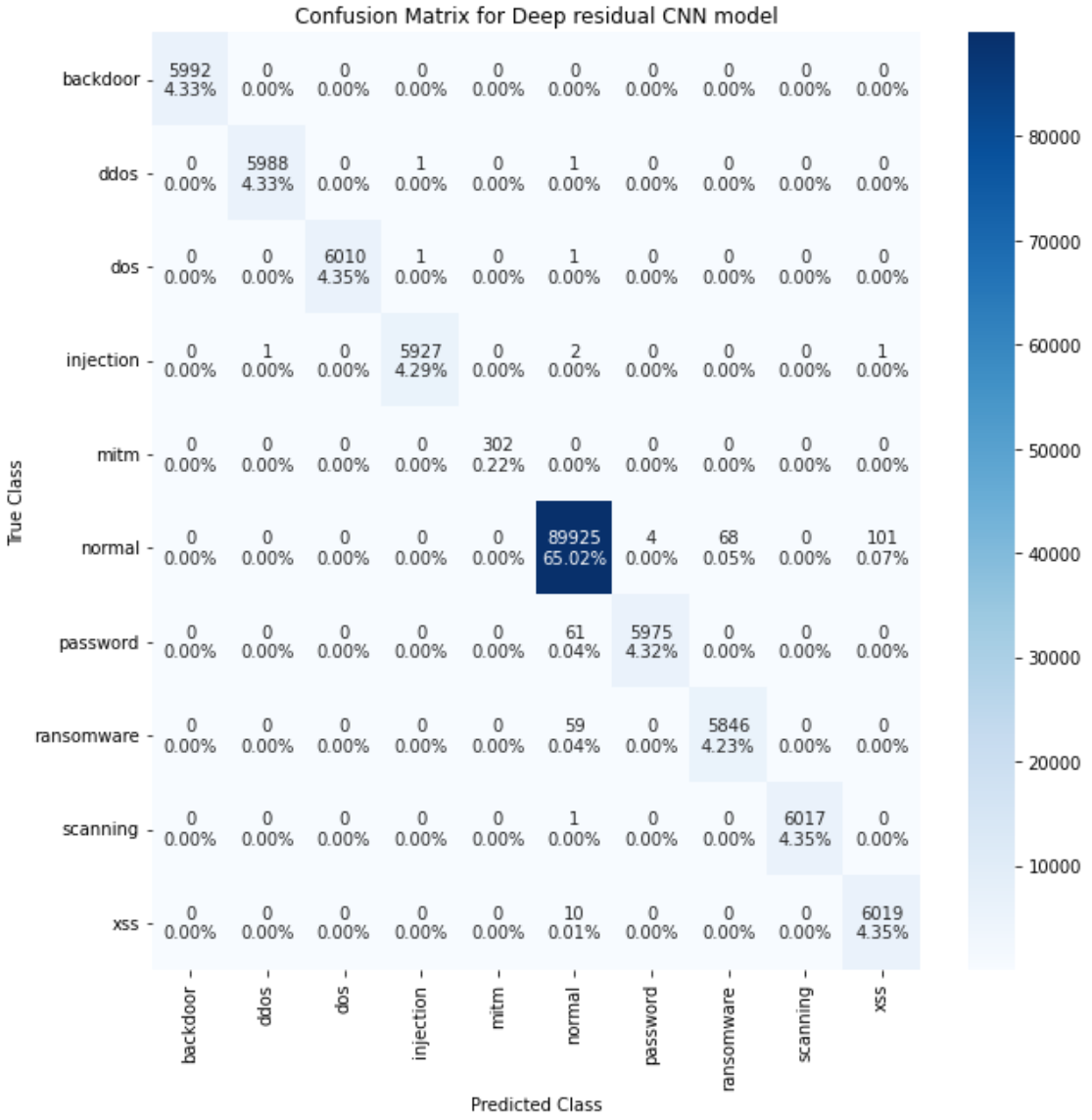
	precision	recall	f1-score	support
backdoor	0.9998	1.0000	0.9999	5992
ddos	0.9997	0.9988	0.9992	5990
dos	0.9998	0.9990	0.9994	6012
injection	0.9993	0.9995	0.9994	5931
mitm	1.0000	1.0000	1.0000	302
normal	0.9975	0.9971	0.9973	90098
password	0.9987	0.9892	0.9939	6036
ransomware	0.9882	0.9897	0.9889	5905
scanning	0.9995	1.0000	0.9998	6018
xss	0.9712	0.9844	0.9778	6029

```
    accuracy                           0.9964    138313
   macro avg     0.9954    0.9958    0.9956    138313
weighted avg     0.9964    0.9964    0.9964    138313

4323/4323 [==============================] - 44s 10ms/step - loss: 2.3045e-04 -
accuracy: 0.9964
Test: accuracy = 0.996428  ;  loss = 0.000230
```



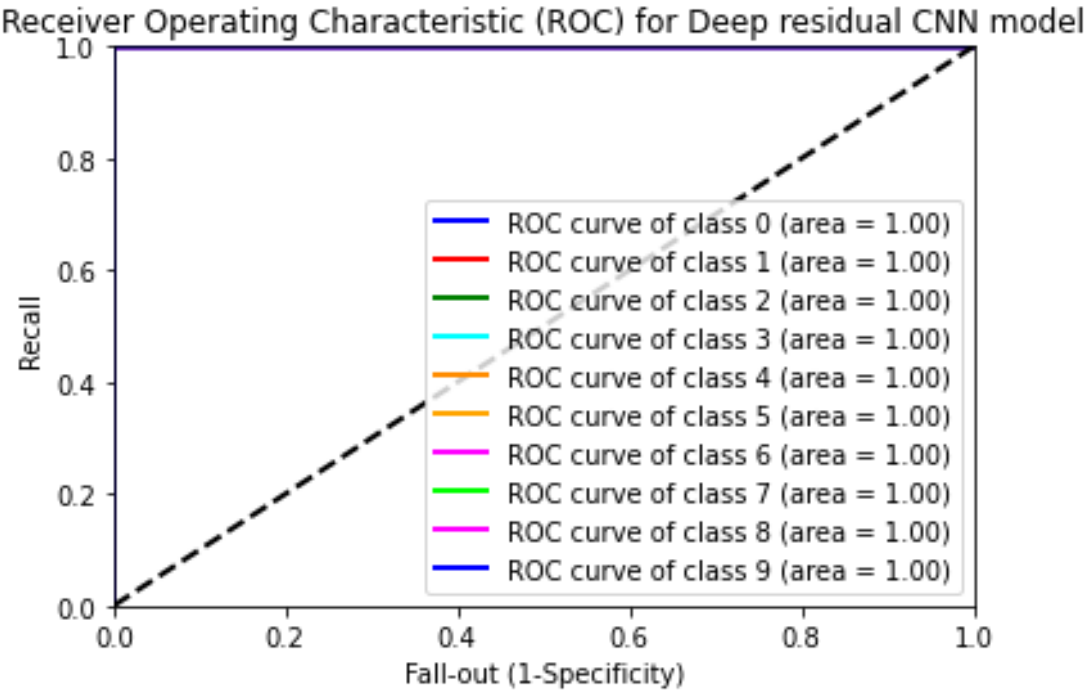
 # Evaluation Metrics of Deep residual CNN model

eval_metrics(model2, model2Name)



	precision	recall	f1-score	support
backdoor	1.0000	1.0000	1.0000	5992
ddos	0.9998	0.9997	0.9997	5990
dos	1.0000	0.9997	0.9998	6012
injection	0.9997	0.9993	0.9995	5931
mitm	1.0000	1.0000	1.0000	302
normal	0.9985	0.9981	0.9983	90098
password	0.9993	0.9899	0.9946	6036
ransomware	0.9885	0.9900	0.9893	5905
scanning	1.0000	0.9998	0.9999	6018
xss	0.9833	0.9983	0.9908	6029
accuracy			0.9977	138313
macro avg	0.9969	0.9975	0.9972	138313
weighted avg	0.9978	0.9977	0.9977	138313

```
4323/4323 [==============================] - 30s 7ms/step - loss: 1.3823e-04 -
accuracy: 0.9977
Test: accuracy = 0.997744  ;  loss = 0.000138
```



Results

####The overall accuracy and loss calculated while testing the model:

1. CNN-LSTM Model

- Accuracy - 99.64%
- Loss - 0.023%

2. DRCNN Model

- Accuracy - 99.77%
- Loss - 0.013%