Received: 10 May 2021 | Revised: 28 July 2021 | Accepted: 29 December 2021

CAAI Transactions on Intelligence Technology

DOI: 10.1049/cit2.12078

ORIGINAL RESEARCH

Network anomaly detection using deep learning techniques

Mohammad Kazim Hooshmand | **Doreswamy Hosahalli**

Department of Computer Science, Mangalore University, Mangalore, India

Correspondence

Mohammad Kazim Hooshmand, Department of Computer Science, Mangalore University, Mangalore, India.
Email: kazimhooshmand@gmail.com

Abstract

Convolutional neural networks (CNNs) are the specific architecture of feed-forward artificial neural networks. It is the de-facto standard for various operations in machine learning and computer vision. To transform this performance towards the task of network anomaly detection in cyber-security, this study proposes a model using one-dimensional CNN architecture. The authors' approach divides network traffic data into transmission control protocol (TCP), user datagram protocol (UDP), and OTHER protocol categories in the first phase, then each category is treated independently. Before training the model, feature selection is performed using the Chi-square technique, and then, over-sampling is conducted using the synthetic minority over-sampling technique to tackle a class imbalance problem. The authors' method yields the weighted average *f*-score 0.85, 0.97, 0.86, and 0.78 for TCP, UDP, OTHER, and ALL categories, respectively. The model is tested on the UNSW-NB15 dataset.

KEYWORDS

artificial intelligence, convolution, neural network, security

1 | INTRODUCTION

Cyber-attacks are continuously evolving with the sophistication and advancement of hardware, software, and network topologies. Intrusion detection systems are highly recommended in defending the network from malicious cyber-attacks [1]. Recent studies show that deep learning (DL) techniques are used in various areas such as natural language processing (NLP), speech recognition, computer vision, cyber-security etc., because they have the capabilities of handling high-dimensional data with imbalanced classes and non-linear properties [2–4]. A dataset is called imbalanced if a sample of one class contains more instances than the others [5]. An imbalanced dataset can be binary-class or multi-class. However, many conventional machine learning (ML) algorithms such as support vector machine (SVM), Naive Bayes (NB), decision tree (DT), random forest (RF), and many more are proposed by the previous studies for network anomaly detection [4, 6–10], but the main limitation is that to evaluate the model performance, only well-balanced network traffic data is adopted. Most of the standard algorithms lead to a biased result towards the majority class in an imbalanced dataset because they tend to favour the majority class over the minority class. In such a condition, they will yield higher results for the majority classes and lower results for the minority ones [11, 12]. To overcome this limitation and to tackle non-linear properties of datasets, the development of DL models with a combination of sampling techniques are encouraged. To resolve an imbalance issue, there are many sampling techniques. One of the most commonly used of such techniques is the synthetic minority over-sampling technique (SMOTE), which is proposed by Chawla [13]. The SMOTE method generates additional synthetic samples from the minority class to balance the data. In this study, 1-D convolutional neural network (CNN) architecture is trained on a sample of the newly generated UNSW-NB15 dataset. From feature visualisation of the UNSW-NB15 dataset in Figure 1, we observe that the data distribution is different from protocol to protocol, overlap of classes and distribution are completely different. This difference will impact the classifier's behaviour so that we split the data into different protocol categories first and then treated each one separately. On the other hand from Table 1, we can see that the data is very imbalanced, for example, 'worms', 'shellcode', 'backdoor', and 'analysis' are very less in terms of the number of instances compared to the other classes in all protocol categories. In

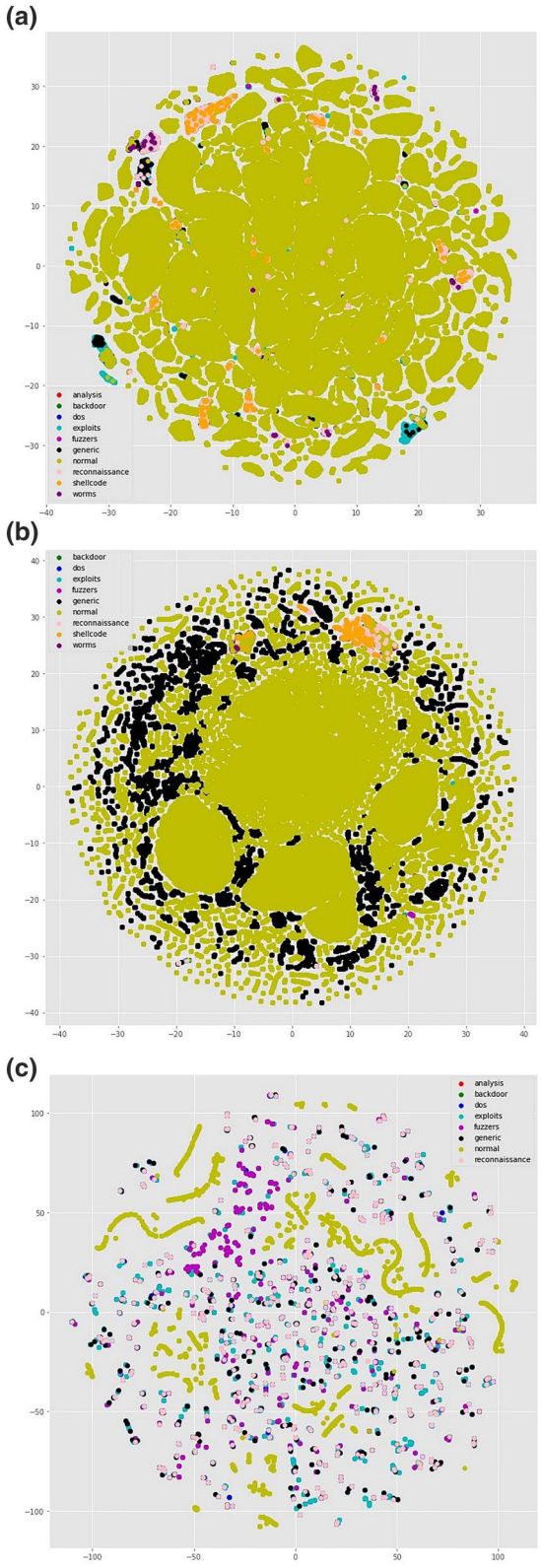


FIGURE 1 *t*-Distributed stochastic neighbour embedding (*t*-SNE) visualisation of full UNSW-NB15 dataset. (a) *t*-SNE visualisation of transmission control protocol; (b) *t*-SNE visualisation of user datagram protocol; (c) *t*-SNE visualisation of OTHER

TABLE 1 Distribution of classes over different protocols in UNSW-NB15

No	Class	TCP	UDP	OTHERS	ALL
1	normal	1,436,890	766,685	15,189	2,218,764
2	generic	3118	210,600	1763	215,481
3	exploits	27,443	874	16,208	44,525
4	fuzzers	15,474	6043	2729	24,246
5	dos	3336	527	12,490	16,353
6	reconnaissance	6965	4890	2132	13,987
7	analysis	622	-	2055	2677
8	backdoor	323	34	1972	2329
9	shellcode	750	761	-	1511
10	worms	153	21	-	174
	Total	**1,495,074**	**990,435**	**54,538**	**2,540,047**

Abbreviations: TCP, transmission control protocol; UDP, user datagram protocol.

addition to that, some classes do not exist under some protocols, for example, 'analysis' does not exist under user datagram protocol (UDP), and 'worms' and 'shellcode' do not exist under the 'OTHER' category. To tackle the issues of imbalance, we applied the SMOTE over-sampling technique on minority classes for each protocol category separately. The main steps that we followed in this study are data pre-processing, data division into protocols, feature selection using the Chi-square technique, over-sampling of minority classes using SMOTE technique, and applying 1-D CNN for multi-classification.

Overall, this study has made the following contributions in the field of cyber-security:

- By combining the SMOTE over-sampling technique and the 1-D CNN technique, the proposed model is able to classify minority classes of attacks with a higher accuracy rate.
- Features visualisation is conducted using a *t*-distributed stochastic neighbour embedding (*t*-SNE) method prior to training the model. From Figure 1, we know that the distribution of different classes vary from protocol to protocol, so we divided the data into different protocol categories and then performed classification on each category as well as on the total data. We compared the results and observed that the model is performing better in protocol-wise multi-classification compared to all protocols combined.

The rest of this paper is arranged as follows: section 2 presents a summarised literature review; section 3 gives dataset descriptions; section 4 provides the proposed method; experimental results are discussed in section 5; result analysis and comparison are provided in section 6, and section 7 describes the conclusion and future work.

2 | RELATED WORKS

In [3], the authors proposed three different CNN architectures (i.e. deep CNN, moderate CNN, and shallow CNN) for network anomaly detection. They conclude that the shallow CNN mostly yields better accuracy compared to the others. In [14], the authors proposed an intrusion detection model using CNN; the performance was evaluated on NSL-KDD (NSL-KDD Train+, NSL-KDD Test+) dataset in TensorFlow. They performed feature extraction and data pre-processing. Categorical features were encoded using a one-hot encoding technique; numeric variables were normalised using standard scalar, and then, the dataset was transformed into binary vectors with 464 variables. In the next step, those binary vectors were used to generate images with 8 * 8 grey-scale pixels, and finally, two CNN models GoogLeNet and ResNet-50 were used to identify the image-conversion category. The experimental results show that GoogLeNet and ResNet-50 scored 90.01% and 89.85% of the highest *f*-score, respectively. The parameters setting of this study is as follows: the number of epochs for both models was 100; the batch size was 62 for GoogLeNet and 256 for ResNet-50; the optimiser was gradient descent; the loss function was set to cross-entropy.

In the work done in [15], deep neural networks were proposed towards the task of anomaly detection. The result shows that DL methods are good for detecting anomalies in the software-defined network.

In [16], the authors employed simple CNN and hybrid CNN models. The simple model had multiple layers while the hybrid model was utilised using Long Short-Term Memory (LSTM) units, recurrent neural networks (RNNs), and gated recurrent units (GRUs) towards the task of network anomaly detection. The architecture includes an input layer, a hidden layer with one or more CNN layers followed by feed-forward networks (FFNs), and an output layer. The result of binary classification using the KDDCup99 dataset for both 1 layer and 2 layers showed a 99% *f*-score while the third layer CNN with LSTMs showed 98.7% accuracy for multi-classification. The model was evaluated using TensorFlow and the parameters used were as follows: inputs were set to 41 * 1; filters were 16, 32, and 64; filter lengths were set to 3 and 5; the learning rate was given in the range of 0.01–0.5; the number of epochs was up to 1000.

In [17], the authors mentioned the usage of accelerated computing platform techniques with the aim of reducing training time and speeding up multi-classification of attacks. In [18], the authors proposed Stacked Auto-Encoders (SAE) for multi-classification of attacks and as well as for deep feature extraction. The result was good compared to existing approaches. In [19], the authors proposed RNN for multi-classification of attacks and the result was promising. They evaluated their approach on a sample of the NSL-KDD dataset.

3 | UNSW-NB15 DATASET

Earlier datasets such as DARPA 98/99, KDDCup99, and NSL-KDD were suffering from certain issues such as: (1) unavailability of knowledge on modern attacks' behaviours, (2) non-similarity of normal traffic existing in those datasets with a new pattern of normal traffic (because those datasets were generated decades ago), and (3) difference in the distribution of attack-types on training_set and testing_set [20]. To overcome these issues of previous datasets, the UNSW-NB15 dataset was generated in 2015 [20–22] by Moustafa and Slay in the Australian Center for Cyber-Security lab (ACCS). The UNSW-NB15 has 49 features including 2 labels (binary and multi-class). The attack categories and the classes are described in detail in [21, 23] Table 1.

4 | PROPOSED METHOD

The main steps of our proposed method are (1) data pre-processing, (2) dividing data into protocol categories such as transmission control protocol (TCP), UDP, OTHER, and ALL, (3) sampling (over-sampling the minority class and under-sampling the majority class), and (4) 1-D CNN-based classification. Figure 3 describes the architecture of our method.

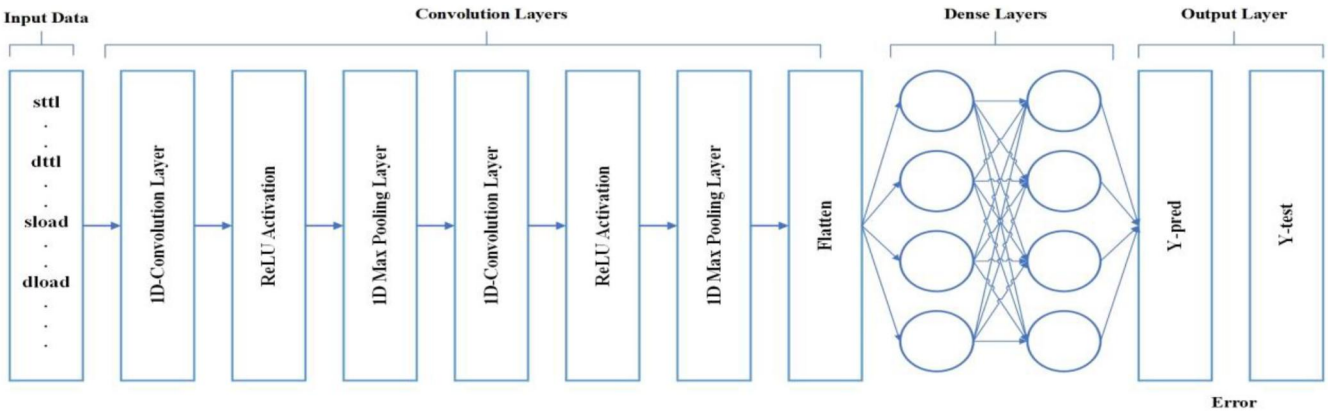


FIGURE 2 General architecture of 1-D Convolutional neural network

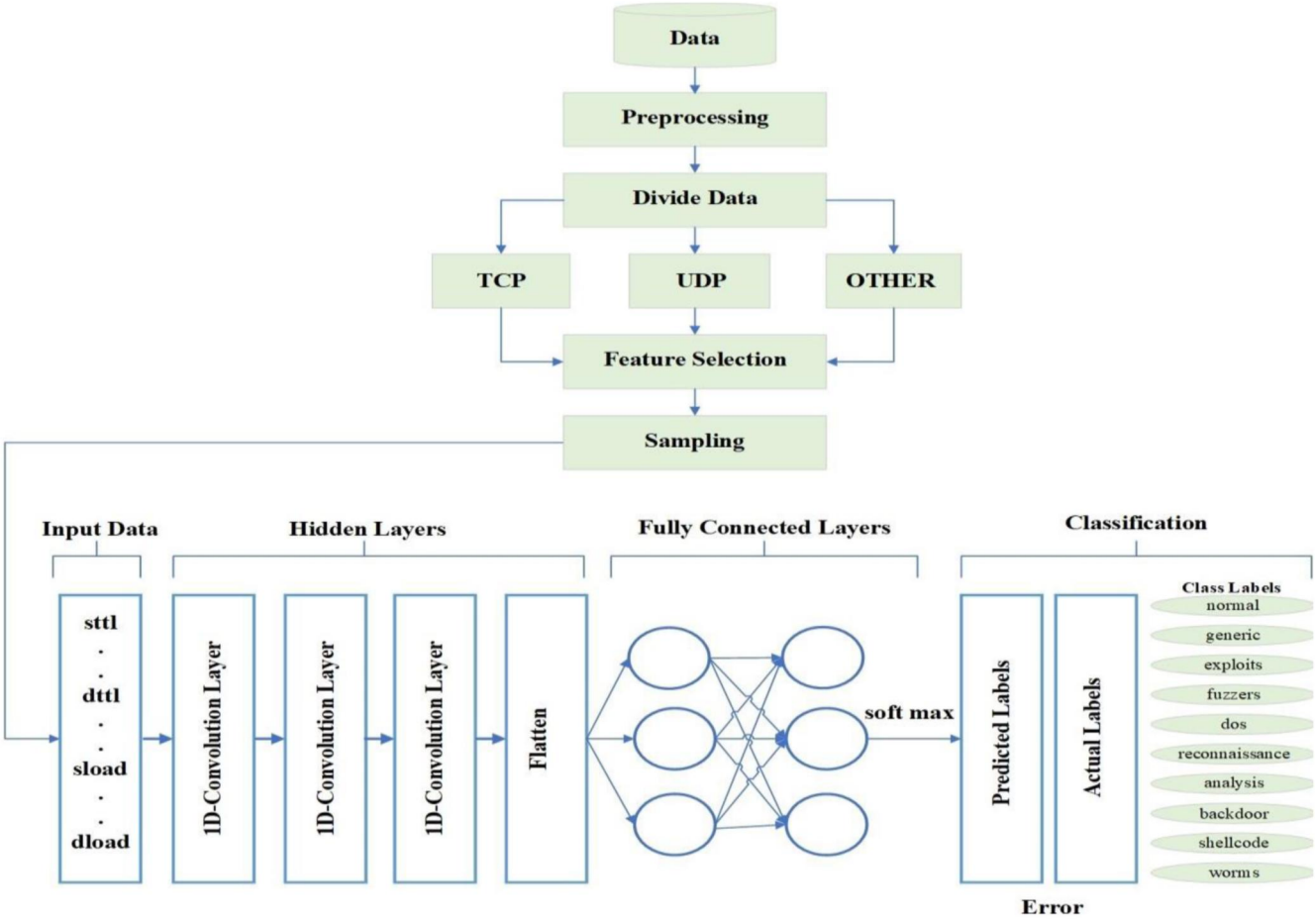


FIGURE 3 Architecture of the proposed model

4.1 | Data pre-processing

Switch information ('srcip', 'sport', 'dstip', 'dsport') and time-related features ('ltime' and 'stime') are eliminated from the dataset because they are redundant features. Categorical features ('proto', 'state', and 'service') have many values each. To encode these features, we apply a one-hot encoding technique, which will add a new variable for each value in each categorical feature. It will make the process very complex because it will add hundreds of new features to the data. To avoid that complexity, we categorise categorical values in such a way that feature values with more number of instances are kept as it is, and the remaining feature values with a lesser number of instances are put collectively under the 'OTHER' category. Reference to Table 2, we decided which feature value has to be kept as it is and which one has to be put under OTHER category. From the following categorization we can see that 'state' has FIN (finish), CON (connected), and INT (initiation). Similarly 'service' has '-' and domain name server (DNS).

- proto: 'TCP', 'UDP', 'OTHER'.
- state: 'FIN', 'CON', 'INT', 'OTHER'.
- service: '-', 'DNS', 'OTHER'.

TABLE 2 Categorical features

Categorical features	Type	Count	Percentage
Proto	TCP	1,495,074	58.9%
	UDP	990,435	39.0%
	OTHER	54,538	2.1%
State	FIN	1,478,689	58.2%
	CON	560,588	22.1%
	INT	490,471	19.3%
	OTHER	10,299	0.4%
Service	-	1,246,397	49.1%
	DNS	781,668	30.8%
	OTHER	511,982	20.2%

Abbreviations: DNS, domain name server; TCP, transmission control protocol; UDP, user datagram protocol.

There are a few null values and spelling differences that exist in columns 'ct_ftp_cmd', 'is_ftp_login', and 'ct_flw_http_mthd', which are handled appropriately, and all null values under the 'attack_cat' feature are replaced with 'normal'. The nominal features are encoded and then the

TABLE 3 Selected features of each category

Category	Selected features
TCP	dur, sttl, dttl, sload, dload, smeansz, dmeansz, sjit, sintpkt, dintpkt, tcprtt, synack, ackdat, ct_flw_http_mthd, ct_state_ttl, ct_srv_dst, ct_srv_src, ct_src_ltm, ct_dst_ltm, is_ftp_login, service
UDP	dur, sttl, sbytes, sload, dttl, dload, dpkts, spkts, smeansz, sjit, dmeansz, sintpkt, ct_src_dport_ltm, ct_state_ttl, ct_dst_ltm, state, ct_dst_src_ltm, service
OTHER	sbytes, sttl, sload, dpkts, smeansz, dmeansz, sjit, djit, sintpkt, dintpkt, ct_state_ttl, ct_srv_src, ct_srv_dst, ct_src_ltm, ct_dst_ltm, ct_dst_src_ltm, ct_dst_sport_ltm, is_sm_ips_ports, ct_src_dport_ltm, state
ALL	sttl, dload, dttl, sload, smeansz, sintpkt, dmeansz, dintpkt, tcprtt, ackdat, synack, ct_state_ttl, ct_srv_src, ct_dst_ltm, ct_srv_dst, is_sm_ips_ports, proto, state, service

Abbreviations: TCP, transmission control protocol; UDP, user datagram protocol.

normalisation is applied using the min_Max scaling technique. Equation (1) describes the way in which the min_Max method transforms values in the range of 0–1.

$$X = \frac{X_i - X_{\text{minimum}}}{X_{\text{maximum}} - X_{\text{minimum}}}, \quad (1)$$

where X stands for the normalised data, X_i represents the original feature value, X_{minimum} is the minimum-value in feature, and X_{maximum} is the maximum-value in feature before the normalisation.

4.2 | Data division

Prior to feature selection, we split the dataset based on different protocols. From Table 2, we know that TCP protocol contributes about 59%, UDP protocol covers about 39%, and the remaining protocols contribute about 2% of the dataset size. So, we split the data into three parts and treat each part as an independent data hereafter. The reason for dividing data into protocol-wise categories is that different protocol formats require a different set of features subsequently.

4.3 | Feature selection

To eliminate less important features from a dataset, feature selection is necessary. We reduced number of features of all categories (TCP, UDP, OTHER, and ALL) to almost 50% using the Chi-square selection technique. The 'ALL' category is a combination of TCP, UDP, and OTHER after under-sampling the majority class. Under-sampling and over-sampling are explained in Section 4.4.

Detailed information on UNSW-NB15 and its categories of features is given in [23]. Table 3 shows the list of selected features for each category.

TABLE 4 Sample size of each protocol category

Class	TCP	UDP	OTHER	ALL
normal	21,553	23,001	9113	53,667
fuzzers	5014	5004	2729	12,747
reconnaissance	4855	4890	2132	11,877
exploits	5214	874	5024	11,112
generic	3118	5054	1763	9935
dos	3336	527	4746	8609
analysis	622	NA	2055	2677
backdoor	323	34	1972	2329
shellcode	750	761	NA	1511
worms	153	21	NA	174

Abbreviations: TCP, transmission control protocol; UDP, user datagram protocol.

4.4 | Sampling

Recently, different methods are proposed to solve the issue of class imbalance in a dataset. The methods are broadly grouped under two categories: algorithm level and data level [24]. An algorithm-level approach works on modifying ML algorithms to accommodate the imbalanced dataset. Examples of such modifications can be cost-sensitive methods/cost error minimisation rather than accuracy rate maximisation [25]. The data-level techniques are trying to re-balance the data class distribution either by decreasing the majority class or increasing the minority class. Various sampling methods exist, some of the examples can be random under-sampling, random over-sampling, direct over-sampling, SMOTE, and many others [24]. Over-sampling techniques add some more instances to the minority class in the training set, the simplest method is random over-sampling [26] but the drawback is over-fitting [27]. To overcome this problem, Chawla [13] proposed the SMOTE technique, which generates synthetic samples from the minority class. Samples created by the SMOTE technique are a linear combination of

two identical samples from the minority class [13, 26, 28]. The SMOTE over-sampling algorithm works as follows: Let S represent the size of a small class, considering a sample j of the small size class, and x_j denotes its feature vector such that, $j \in \{1, ..., S\}$:

- Find k neighbours of the sample x_j from all S (using the Euclidean Distance for example) and denoted it as $x_{j(near)}$, $near \in \{1, ..., k\}$.
- $x_{j(nn)}$ sample is selected randomly from the k neighbours, and the random number β_1 between 0 and 1 is generated to synthesise a new sample x_{j1} as in the equation $x_{j1} = x_j + \beta_1 * (x_{j(nn)} - x_j)$.
- Step 2 is repeated M times to synthesise M new samples: $x_{j(new)}$, $new \in \{1, ..., M\}$.

In our case, we are under-sampling a few majority classes using random under-sampling. Then, we applied SMOTE over-sampling approach to over-sample the minority classes in the training set. Table 4 provides the size of each class sample with respect to protocol categories.

4.5 | 1-D CNN

CNNs are the extension of feed-forward neural networks (FFNNs). The main components of CNNs are convolutional layer, non-linear activation function (e.g. Rectifier Linear Unit [*ReLU*]), and pooling layer (s) followed by one or more fully connected layers for the classification [29]. Figure 2 illustrates the general architecture of 1-D CNN.

- Convolutional Layer: Its main task is to extract feature maps from vector input data using filters followed by activation functions such as ReLU, tanh etc. [16]. The feature map calculation using *ReLU* and tanh non-linear functions are given in Equations (2) and (3), respectively.

$$H_a^k = \max\left(w^k x_a, 0\right) \tag{2}$$

$$H_a^k = \tanh\left(w^k x_a\right), \tag{3}$$

where H^k stands for *kth* feature map, a represents an index in the feature map, x_a stands for the input, and w^k represents the weights. In this study, we used the *ReLU* activation function.

- Pooling Layer: Its main task is to under-sample each feature map's dimensions but keep the most informative feature map out of them. Hence, it reduces the computational cost and controls over-fitting in the model. The two commonly used pooling layers are average pooling and max pooling [30]. In the max pooling, only the largest value from the window is selected by the function while the average pooling function considers the mean of all values in the window of the array currently covered by the kernel. The calculation methods of max pooling and average pooling are described in Equations (4) and (5), respectively.

$$f_{\max}(x) = \max(x_a) \tag{4}$$

$$f_{\text{avg}}(x) = \frac{1}{n}\sum_{a=1}^{n}(x_a), \tag{5}$$

where x describes the vector of input data with an activation function, and n denotes a local pooling region. A detailed information on the pooling concept is given in [31, 32].

- Fully Connected Layer: As the word itself describes, the fully connected means that every neuron of two consecutive layers are connected to each other. It is a multi-layer perceptron (MLP) that employs the *softmax* function in the output layer. A fully connected layer is doing the same task that the standard artificial neural networks are doing, that is, to generate class scores from activations, which will be used for the classification [31].
- Activation Layer: It is the last layer of CNNs architecture that works similar to activation functions of the other neural networks. Some of the common activation functions employed in CNNs are *ReLU*, *sigmoid*, and *softmax*. Our model uses a *softmax* activation function for the classification problem of multi-class. The *softmax* function calculates the probabilities of each class and selects the highest value among the range of probability to produce the most accurate value for a class. The mathematical calculation of *softmax* is described in Equation (6).

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n}(e^{x_j})}, \tag{6}$$

where x denotes an input. It applies the standard exponential function to each element x_i of the input vector x and normalises these values by dividing by the sum of all these exponentials. Connecting all the above-mentioned points, the convolutional layer and pooling layer perform feature extraction duties while the fully connected layer is doing classification tasks. Figure 3 shows all the main steps of our proposed method, the CNN model contains three 1-D layers with a *ReLU* activation function followed by a flatten layer. The classification is done using two fully connected layers using *softmax* activation.

5 | EXPERIMENTAL RESULTS

The hardware and software configurations for our experiments are Windows 10 with an Intel®Core™ i7-9700 central processing unit running at 3.00 GHz, 8 processors, 16 GB RAM, and 1 TB hard disk. Python 3.6 with

libraries of *scikit learn* and *Keras* [33, 34] was utilised. As CNN is a parametrised function, so its performance fully depends on the optimal parameters, some of the parameters are the number of epochs, the number of hidden layers, the number of nodes in each layer etc. In [16], the authors claim that the lower learning rates have shown better results in differentiating the connection records using CNN. It is also mentioned that to attain the acceptable detection rate, the lower learning rate anticipate more number of epochs for lower frequency classes. In [35], the learning rate was assigned to 0.01 and the *adam* optimiser was utilised for CNN architecture. In [1], the authors claim that, in the case of multi-classification, the *ReLU* activation function was performing better than *sigmoid* and tanh for 500 epochs. We have conducted different parameter adjustments in a tuning phase. The number of epochs was tested for 100–500, we decided on 500 because of its higher performance for all the categories of data (TCP, UDP, OTHER). The *optimiser* is set to 'adam' with default parameters, the *loss* is set to 'categorical_crossentropy', and the *metrics* are set to *accuracy*, the batch size is 1, the number of nodes in each layer is 512, and the activation function is *ReLU*.

The required time for running our model on the given configuration is calculated as follows: by calculating the required time for a single epoch execution and the number of epochs, we found that it takes approximately 7s to run one epoch. In the proposed method, we have considered 500 epochs, so the total time complexity of the model is 3500s. As per big *O* notations with respect to the number of loops and iterative approach, the complexity was $O(n)$.

The metrics we used are Accuracy, Precision, Recall, and *f*-score. We used an accuracy measure to see the overall prediction of our model; it is good if you have a systematic dataset (false negatives false positives counts are close). Since our data has uneven class distribution, so accuracy will not give an accurate idea on every class, so we have used an *f*-score as well, which is best for uneven class data. Precision is used to be confident of our true positive value and Recall is used to get an understanding of the sensitivity of our prediction. All these metrics are explained and formulated in the Equations (7)–(10).

Accuracy: shows the ratio of truly detection over total transactions and calculated as

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}. \tag{7}$$

Precision: shows how many predicted intrusions by NIDs are actual intrusions and calculated as

$$\text{Precision} = \frac{TP}{(TP + FP)}. \tag{8}$$

Recall: it is the ratio of predicted intrusions versus all intrusions presented and calculated as

$$\text{Recall} = \frac{TP}{(TP + FN)}. \tag{9}$$

F-score: is a harmonic mean of Recall and Precision and gives a better measure of the accuracy and calculated as

$$f1_{\text{score}} = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}, \tag{10}$$

where true positive is denoted by TP, false positive is shown by FP, TN is true negative, and FN is false negative.

Table 5 represents the confusion matrix of 'TCP' data, Table 6 shows the confusion matrix for 'UDP', and similarly, Tables 7 and 8 provide the confusion matrix for 'OTHER' and 'ALL', respectively. From these confusion matrixes, we know that the accuracy for 'ALL' is lower for 'backdoor', 'dos', 'exploits', and 'fuzzers' compared to all the three categories (TCP, UDP, and OTHER). However, for 'shellcode' and 'analysis', the accuracy is higher for 'ALL', and for the remaining classes, it is varying. Figure 4 provides a Precision score comparison for all categories. Figures 5 and 6 show a comparison of Recall and *f*-score, respectively. From Figure 4, we can see that the weighted average of precision for 'ALL' is 0.90, which is higher compared to precision of 'TCP' and 'OTHER' but lower than the precision of 'UDP' by 7%. From Figure 5, we can see that the weighted average of Recall for 'ALL' is the lowest among all. As the weighted average of Recall score for 'ALL' is 0.76, which is lower by 20% from the leading score that is, 'UDP', and lower by 10% and 8% from 'OTHER' and 'TCP', respectively. Similarly from Figure 6, we observe that the 'ALL' category scores the lowest among all in terms of *f*-score. Again the leading score belongs to 'UDP' with 0.97 followed by 0.86 and 0.85 for 'OTHER' and 'TCP' categories, respectively. Based on the comparison on Precision, Recall, and *f*-score, we can see that the score of 'ALL' is the lowest among all except for precision, which is higher than the score of 'OTHER' and 'TCP'. It hints that treating protocol-wise lead to better results.

6 | RESULT ANALYSIS AND COMPARISON

6.1 | Comparison

We have compared our method with two recent studies; Table 9 shows an overall performance comparison with the studies done in [1], for multi-classification in terms of accuracy, precision, recall, and f-score. We can see that our approach yields better results in terms of all the metrics.

Meanwhile, we compared our method with the study conducted in [36] for each class; the comparison is shown in Table 10. We observe that for the attack classes, our approach is performing much better except for the exploits and generic. However, in the case of normal, our method gets a lower result with a small percentage of 0.007.

TABLE 5 Confusion matrix for transmission control protocol

Predicted Actual	analysis	Backdoor	Dos	exploits	fuzzers	generic	normal	reconnaisance	shellcode	worms
analysis	176 94%	0	9	2	0	0	0	0	0	0
backdoor	15	52 54%	4	0	2	10	0	6	6	2
dos	254	5	322 32%	284	30	45	1	19	33	8
exploits	284	8	90	947 61%	31	62	1	97	25	19
fuzzers	71	0	7	3	1390 92%	5	0	10	16	2
generic	228	8	28	63	34	557 60%	1	4	9	3
normal	15	0	1	1	55	0	6391 99%	0	3	0
reconnaisance	66	2	8	10	8	2	0	1338 92%	19	4
shellcode	35	0	0	0	21	0	0	7	162 72%	0
worms	7	1	2	4	0	0	0	0	0	32 70%

TABLE 6 Confusion matrix for user datagram protocol

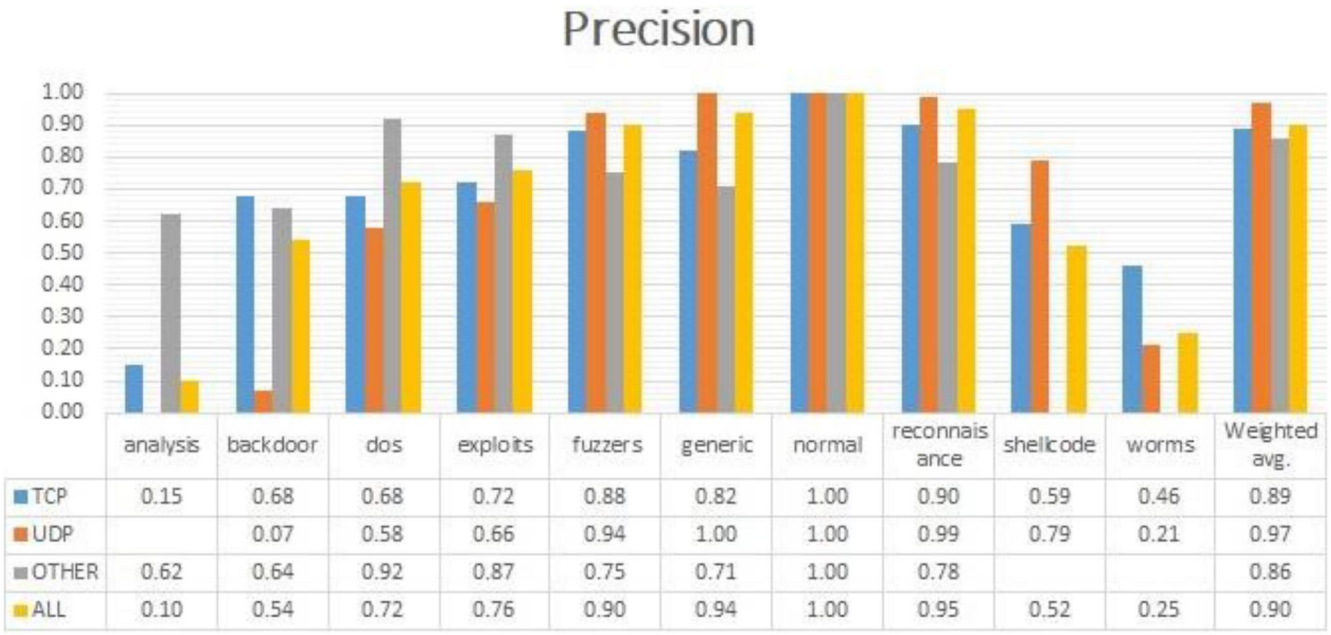
Predicted Actual	backdoor	Dos	exploits	fuzzers	Generic	normal	reconnaisance	shellcode	worms
backdoor	10 100%	0	0	0	0	0	0	0	0
dos	20	78 49%	30	6	0	0	11	8	5
exploits	25	26	177 68%	14	1	0	0	16	3
fuzzers	39	17	58	1358 90%	0	0	2	19	8
generic	1	3	1	0	1510 100%	0	0	0	1
normal	1	4	1	62	0	6833 99%	0	0	0
reconnaisance	3	2	0	0	0	0	1460 100%	2	0
shellcode	38	4	2	7	0	0	6	172 74%	2
worms	0	0	1	0	0	0	0	0	5 83%

TABLE 7 Confusion matrix for OTHER

Predicted / Actual	analysis	Backdoor	dos	exploits	fuzzers	generic	normal	reconnaisance
analysis	300 49%	90	9	47	153	12	0	5
backdoor	25	436 67%	6	17	70	60	0	32
dos	17	9	1296 91%	33	0	21	0	48
exploits	11	43	28	1395 93%	2	23	0	5
fuzzers	46	16	1	3	743 91%	0	0	10
generic	44	56	45	71	7	280 53%	0	22
normal	0	0	0	0	0	0	2733 100%	1
reconnaisance	39	27	23	42	10	50	0	449 70%

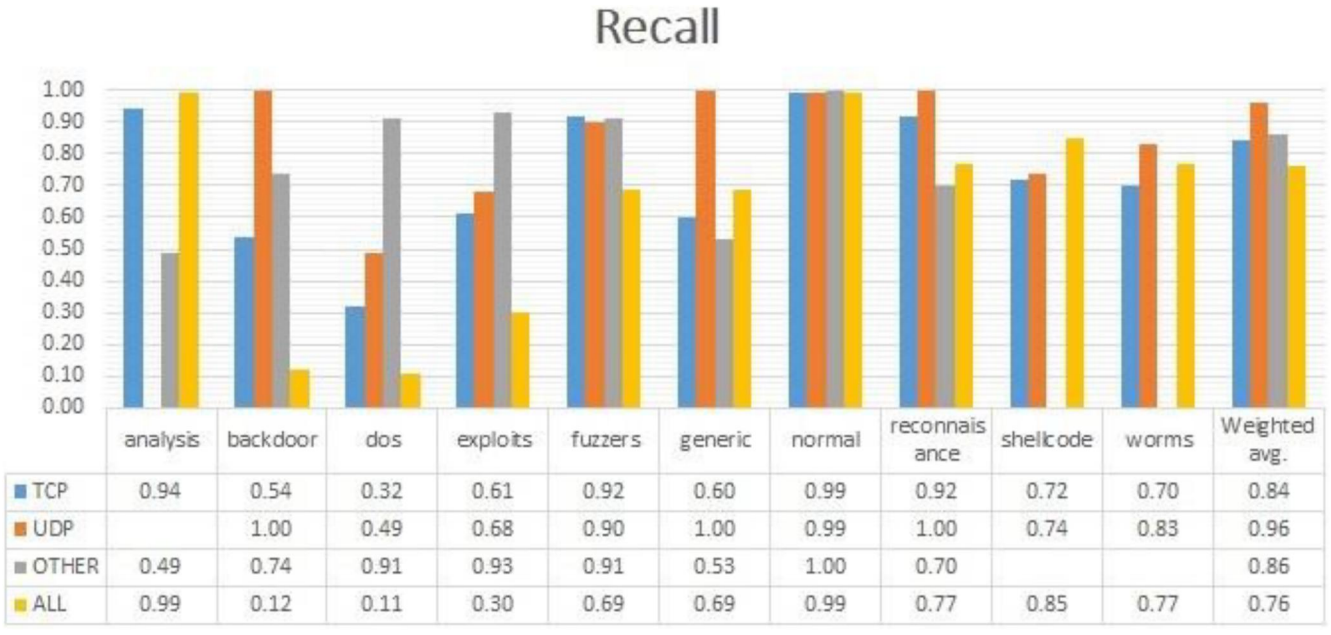
TABLE 8 Confusion matrix for ALL

Predicted / Actual	analysis	backdoor	dos	exploits	fuzzers	generic	normal	reconnaisance	shellcode	worms
analysis	795 99%	0	7	0	1	0	0	0	0	0
backdoor	593	84 12%	2	0	3	2	0	5	10	0
dos	1801	18	272 11%	242	44	59	3	32	88	24
exploits	1934	20	55	1000 30%	55	76	2	73	70	49
fuzzers	986	3	0	14	2657 69%	2	1	19	122	20
generic	735	26	32	50	37	2059 69%	1	6	28	7
normal	43	0	1	1	109	0	15938 99%	1	7	0
reconnaisance	750	3	7	3	8	0	0	2749 77%	26	17
shellcode	18	0	0	0	32	0	0	17	385 85%	1
worms	7	1	0	0	1	2	0	0	1	40 77%



	analysis	backdoor	dos	exploits	fuzzers	generic	normal	reconnaissance	shellcode	worms	Weighted avg.
TCP	0.15	0.68	0.68	0.72	0.88	0.82	1.00	0.90	0.59	0.46	0.89
UDP		0.07	0.58	0.66	0.94	1.00	1.00	0.99	0.79	0.21	0.97
OTHER	0.62	0.64	0.92	0.87	0.75	0.71	1.00	0.78			0.86
ALL	0.10	0.54	0.72	0.76	0.90	0.94	1.00	0.95	0.52	0.25	0.90

FIGURE 4 Precision comparison



	analysis	backdoor	dos	exploits	fuzzers	generic	normal	reconnaissance	shellcode	worms	Weighted avg.
TCP	0.94	0.54	0.32	0.61	0.92	0.60	0.99	0.92	0.72	0.70	0.84
UDP		1.00	0.49	0.68	0.90	1.00	0.99	1.00	0.74	0.83	0.96
OTHER	0.49	0.74	0.91	0.93	0.91	0.53	1.00	0.70			0.86
ALL	0.99	0.12	0.11	0.30	0.69	0.69	0.99	0.77	0.85	0.77	0.76

FIGURE 5 Recall comparison

6.2 | Discussion

To get insight and relate model's performance with the distribution of data, one way is to visualise the data. There are many visualisation techniques like principal component analysis (PCA), *t*-distributed stochastic neighbour embedding (*t*-SNE) etc. In [37, 38], the authors used the *t*-SNE visualisation technique to understand the characteristics of datasets. *t*-SNE is a non-linear dimensionality reduction technique that turns the high-dimensional feature vector into two or more dimensions [1]. We have used *t*-SNE to visualise the output of each layer in our model to understand how each layer is behaving. Figure 7a–d represent the output of the model for TCP, Figure 7e–h visualise the output of the model for UDP; similarly, Figure 8a–d and Figure 8e–h visualise outputs of the model for OTHER and ALL, respectively. From feature visualisation in Figures 7,8, we can see that the overlap of classes in TCP, OTHER and ALL data are much more compared to UDP. UDP

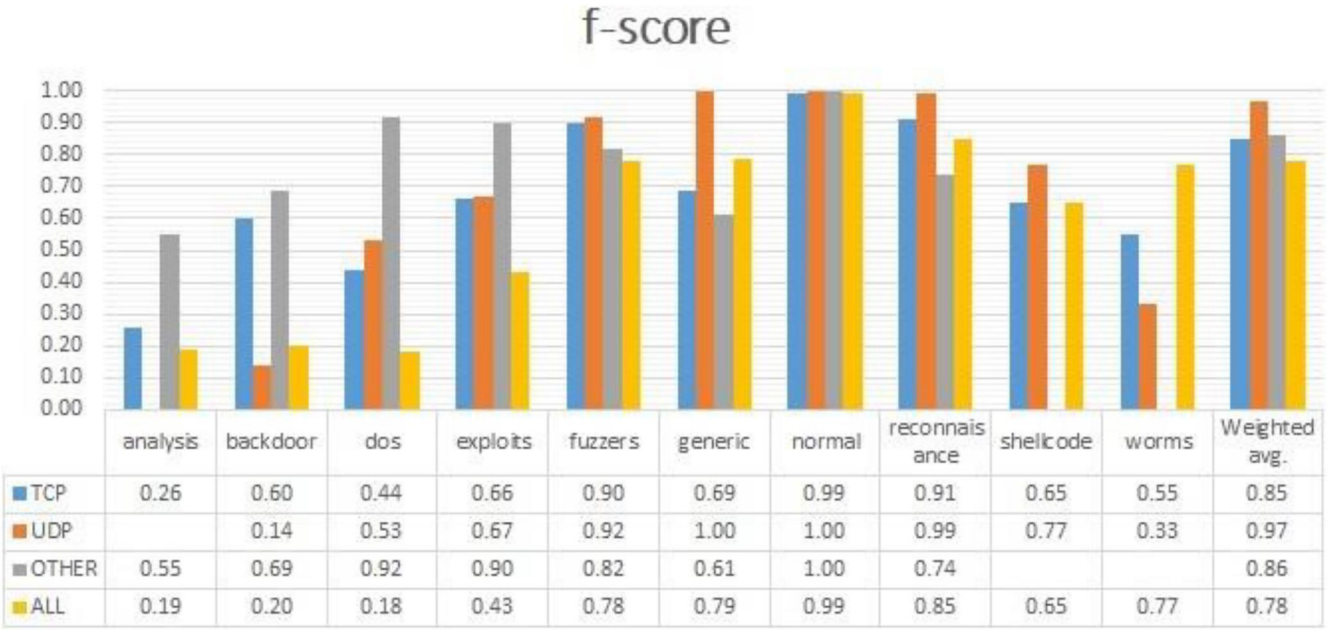


FIGURE 6 *f*-score comparison

TABLE 9 Overall comparison for multi-classification with different metrics

Method	Accuracy	Precision	Recall	*f*-score
Vinayakumar et al. [1]	0.651	0.597	0.651	0.585
Our method	**0.763**	**0.904**	**0.761**	**0.782**

TABLE 10 Class-wise comparison for multi-classification based on accuracy

Method	Potluri et al. [36]	Our method
analysis	0	**0.990**
backdoor	0	**0.120**
dos	0	**0.105**
exploits	0.618	**0.300**
fuzzers	0.068	**0.695**
generic	0.977	**0.691**
normal	0.997	**0.990**
reconnaisance	0	**0.772**
shellcode	0	**0.850**
worms	0	**0.769**

classes are somehow easily separable compared to the other three categories. That might be one of the reasons that the model is performing better in terms of precision, recall, and *f*-score on UDP compared to the other data categories. From the comparison of Figures 4,5,6, we can see that the model weighted scores on the UDP data are 0.97 for precision, 0.96 for recall, and 0.97 for *f*-score, which are the highest among all. We also observe that the performance on the ALL category is lower than TCP, UDP, and OTHER in terms of *f*-score measure, which is why we split the data into different protocol categories, as it was mentioned in section 1.

6.3 | Future work

With the rise of ubiquitous ML and DL, malicious actors are also finding new ways to exploit ML/DL. Adversarial attacks are techniques of such malicious acts, which attempt to fool models with deceptive data and lead to misclassification. Ravi et al. [39] proposed a DL-based framework for the DNS data analysis aiming to detect randomly generated DNS and domain names homograph attacks without the requirement for any reverse engineering or using non-existent domain inspection. The authors conducted detailed experiments to evaluate the robustness of their method on benchmarked datasets against three different adversarial attacks: MaskDGA, Charbot, and DeepDGA. The study highlights the need for more robust detection against adversarial learning. The authors in [40] argue that traditional metrics like, for example, accuracy are not sufficient for evaluating security-sensitive ML systems, so they proposed a method named adversarial robustness score (ARS) for quantifying and comparing adversarial threats for IDSs.

Here, we propose a future framework to extend our current work with an additional feature to evaluate the robustness of our CNN-based model against adversarial attacks. However, network packets impose more constraints on their features that lead to fewer expose to adversarial attacks compared to an image data, but still, the risk is huge and requires cautious treatment.

The main steps of this framework would be

- First, we are going to identify which features are vulnerable for manipulation, for example, inter-arrival time (IAT) and packet length might be more likely manipulable compared to source and destination ports. We will investigate the importance of these features towards prediction if we can leave them out to avoid the risk.
- Identify commonly used adversarial machine learning (AML) based on the literature and evaluate our model's robustness similar to the work done in [40], except that they tested on RNN and we will test on CNN. However, they used their own metric (ARS), we will compare the goodness of ARS against traditional metrics in our scenario.

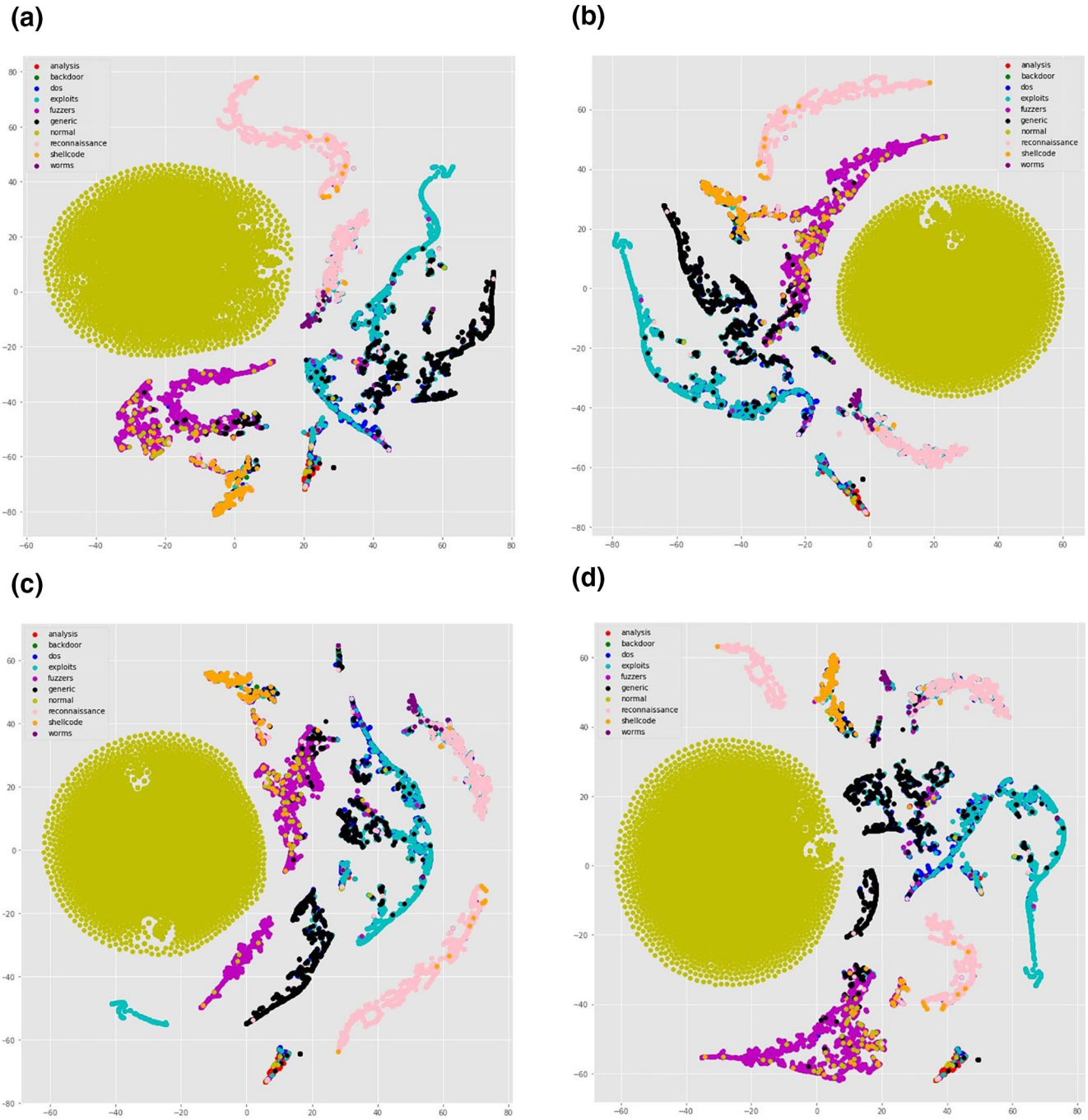


FIGURE 7 *t*-distributed stochastic neighbour embedding (*t*-SNE) visualisation of transmission control protocol (TCP) and user datagram protocol (UDP) in 4 layers of convolutional neural network (CNN). (a) *t*-SNE visualisation of TCP in layer 1; (b) *t*-SNE visualisation of TCP in layer 2; (c) *t*-SNE visualisation of TCP in layer 3; (d) *t*-SNE visualisation of TCP in layer 4; (e) *t*-SNE visualisation of UDP in layer 1; (f) *t*-SNE visualisation of UDP in layer 2; (g) *t*-SNE visualisation of UDP in layer 3; (h) *t*-SNE visualisation of UDP in layer 4

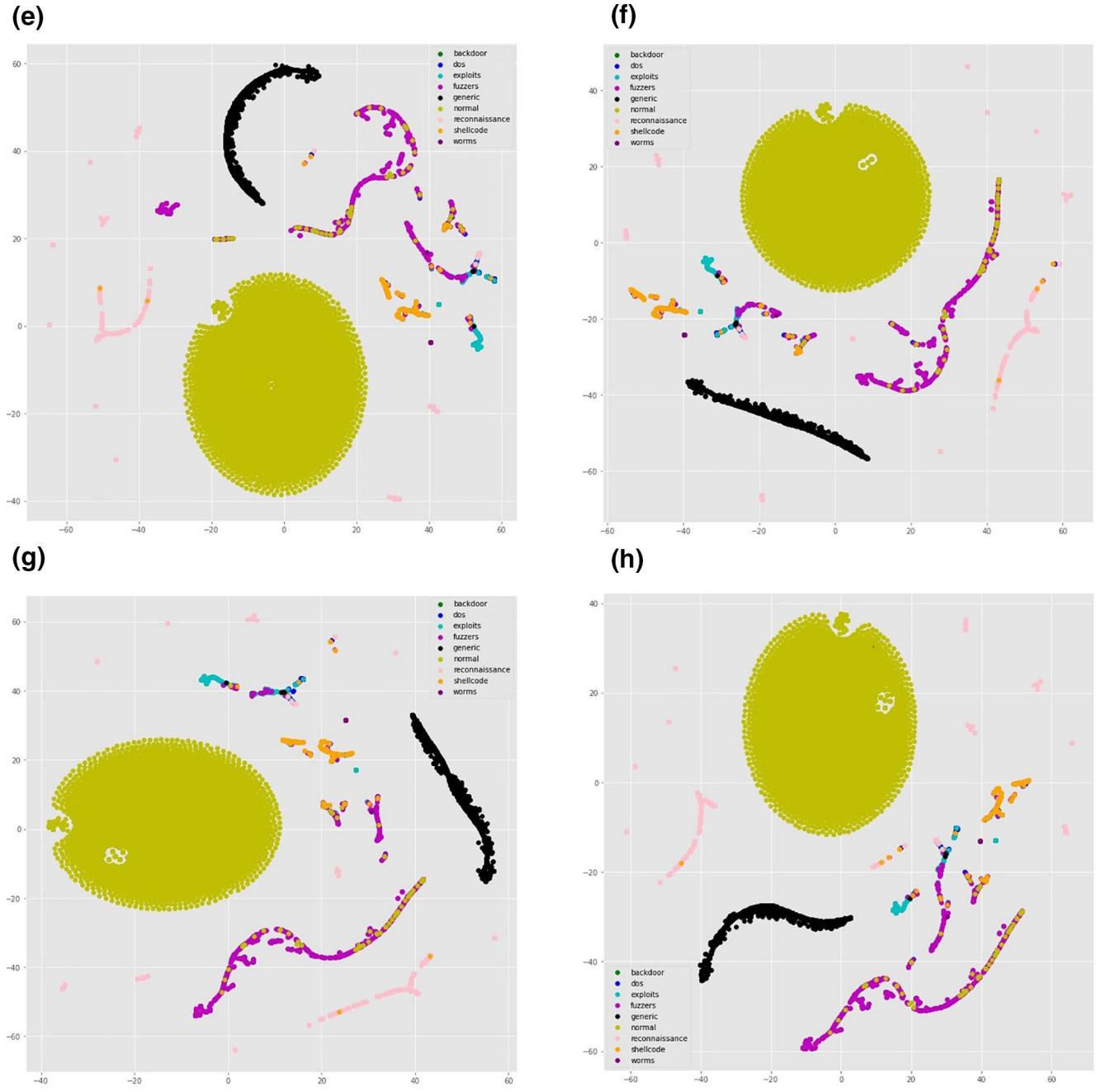


FIGURE 7 (Continued)

- Alternate solution to manipulable feature omission is increasing the robustness of the model against adversarial attacks by augmenting the training set by adversarial attack sample. We will add attack samples and flag them as new attacks and evaluate the model, similar to the work is done in [40], except that we will test on different network architecture and also we will test on a same number of backpropagation as well as on the different number of backpropagation.
- We will evaluate our newly proposed framework on both commonly used datasets that is, UNSW-NB15 and NSL-KDD.

7 | CONCLUSION AND FUTURE WORK

This paper proposes a network anomaly detection using the DL method. We use 1-D CNN architecture to develop a neural

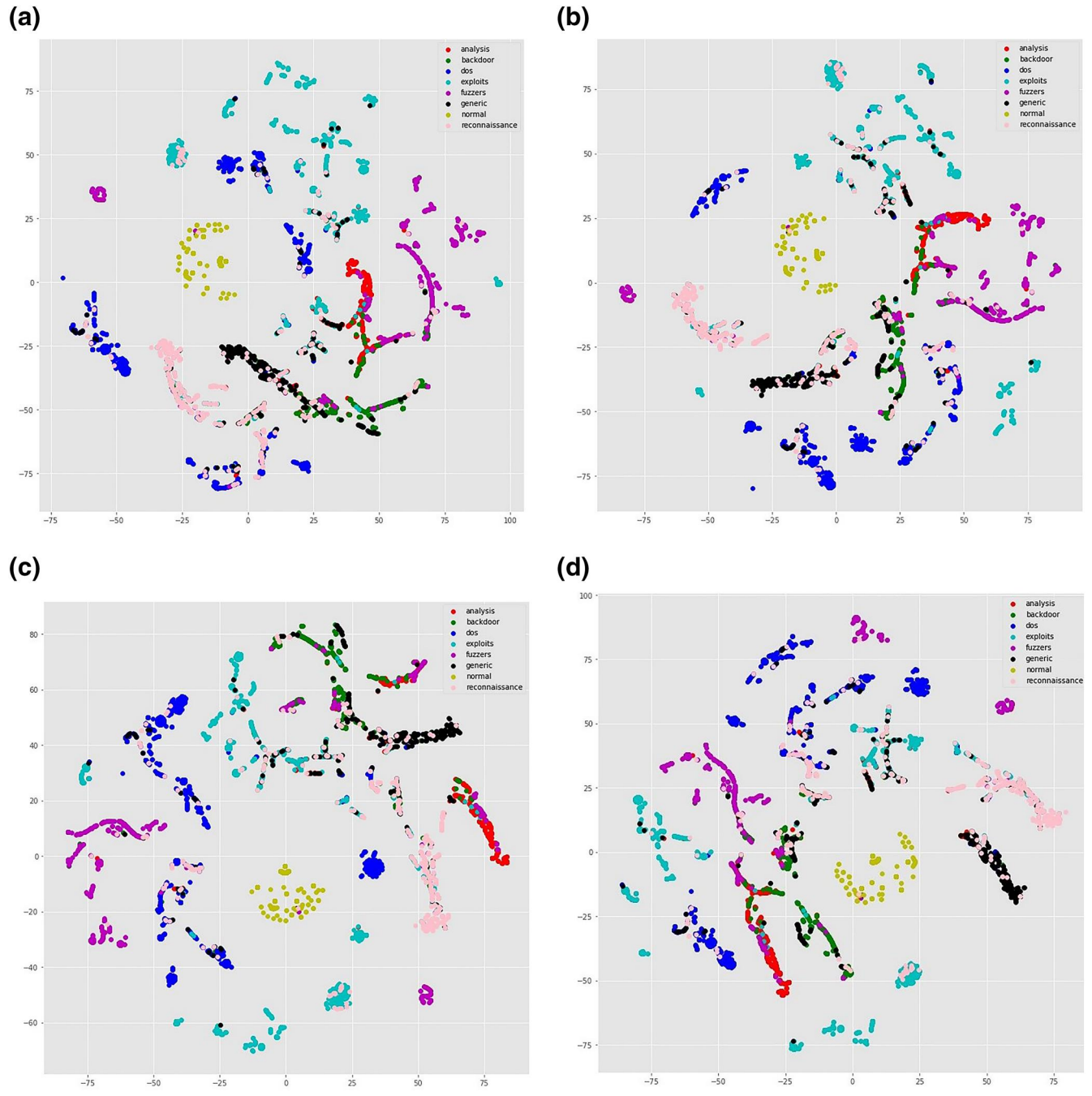


FIGURE 8 *t*-distributed stochastic neighbour embedding (*t*-SNE) visualisation of OTHER and ALL in 4 layers of convolutional neural network (CNN). (a) *t*-SNE visualisation of OTHER in layer 1; (b) *t*-SNE visualisation of OTHER in layer 2; (c) *t*-SNE visualisation of OTHER in layer 3; (d) *t*-SNE visualisation of OTHER in layer 4; (e) *t*-SNE visualisation of ALL in layer 1; (f) *t*-SNE visualisation of ALL in layer 2; (g) *t*-SNE visualisation of ALL in layer 3; (h) *t*-SNE visualisation of ALL in layer 4

network model and incorporate the SMOTE over-sampling method to solve the class-imbalance issue in the dataset. The model was evaluated on the sample of the UNSW-NB15 dataset. The dataset was split into different protocol categories and each category was treated independently. The experimental results show that treating independent categories yield better results in the case of Recall and *f*-score; however, it is not obvious for the Precision. In this study, we used a sample data and applied only one type of over-sampling technique (SMOTE). In the future studies, we will conduct a comparative study on different sampling techniques and test the model with further hyper tuning and full size of benchmarked datasets. Also, we are going to implement the proposed framework suggested in Section 6.3.

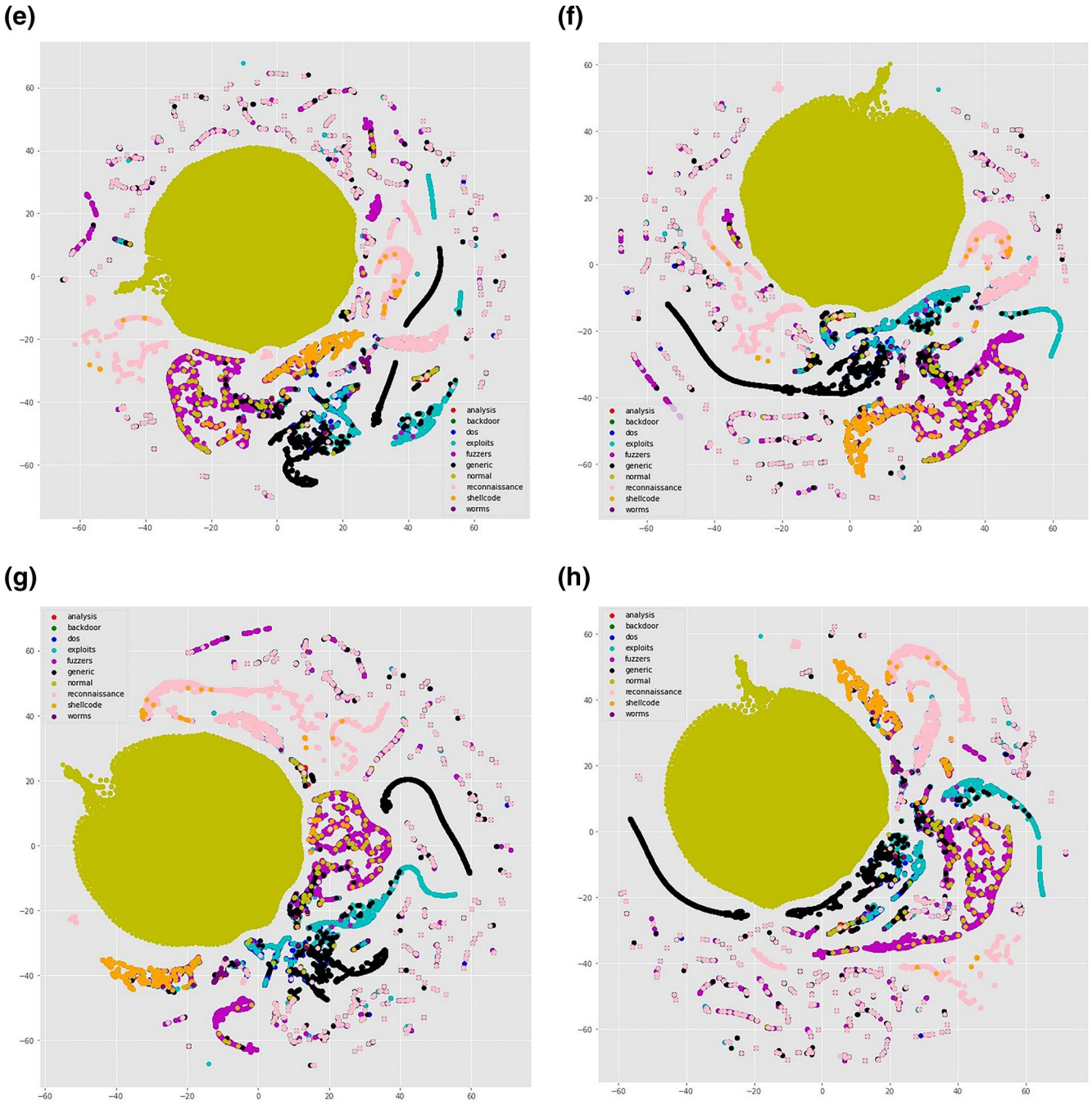


FIGURE 8 (Continued)

ACKNOWLEDGEMENT

None.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

DATA AVAILABILITY STATEMENT

The data has been used in this study is UNSW-NB15, which is freely available for research purposes.

ORCID

Mohammad Kazim Hooshmand https://orcid.org/0000-0002-7840-7383

REFERENCES

1. Vinayakumar, R., et al.: Deep learning approach for intelligent intrusion detection system. IEEE Access. 7, 41525–41550 (2019)
2. Baek, S., et al.: Unsupervised labeling for supervised anomaly detection in enterprise and cloud networks. In: 2017 IEEE 4th International

Conference on Cyber Security and Cloud Computing (CSCloud), pp. 205–210. IEEE (2017)
3. Kwon, D., et al.: An empirical study on network anomaly detection using convolutional neural networks. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1595–1598. IEEE (2018)
4. Deng, L., Yu, D.: Deep learning: methods and applications. Found. Trends Signal Process. 7(3–4), 197–387 (2014)
5. Japkowicz, N., et al.: Learning from imbalanced data sets: a comparison of various strategies. In: AAAI Workshop on Learning From Imbalanced Data Sets, vol. 68, pp. 10–15. AAAI Press, Menlo Park, CA. (2000)
6. Hooshmand M.K., et al.: Using ensemble learning approach to identify rare cyberattacks in network traffic data. In: 2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS), pp. 141–146. IEEE (2020)
7. Primartha, R., Tama, B.A.: Anomaly detection using random forest: a performance revisited. In: 2017 International Conference on Data and Software Engineering (ICoDSE), pp. 1–6. IEEE (2017)
8. Hu, W., Liao, Y., Vemuri, V.R.: Robust anomaly detection using support vector machines. In: Proceedings of the International Conference on Machine Learning, pp. 282–289. Citeseer (2003)
9. Lei, Y.: Network anomaly traffic detection algorithm based on svm. In: 2017 International Conference on Robots & Intelligent System (ICRIS), pp. 217–220. IEEE (2017)
10. Hooshmand, D.M.K.: Machine learning based network anomaly detection. Int. J. Recent Technol. Eng. (2019)
11. Batista, G.E., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newslett. 6(1), 20–29 (2004)
12. Singh, A., Purohit, A.: A survey on methods for solving data imbalance problem for classification. Int. J. Comput. Appl. 127(15), 37–41 (2015)
13. Kumari, C., Abulaish, M., Subbarao, N.: Using smote to deal with class-imbalance problem in bioactivity data to predict mtor inhibitors. SN Comput. Sci. 1, 1–7 (2020)
14. Li, Z., et al.: Intrusion detection using convolutional neural networks for representation learning. In: International Conference on Neural Information Processing, pp. 858–866. Springer (2017)
15. Tang, T.A., et al.: Deep learning approach for network intrusion detection in software defined networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), pp. 258–263. IEEE (2016)
16. Vinayakumar, R., Soman, K., Poornachandran, P.: Applying convolutional neural network for network intrusion detection. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1222–1228. IEEE (2017)
17. Potluri, S., Diedrich, C.: 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. IEEE (2016)
18. Potluri, S., Diedrich, C.: Deep feature extraction for multi-class intrusion detection in industrial control systems. Int. J. Comput. Theory Eng. 9(5), 374–379 (2017)
19. Sheikhan, M., Jadidi, Z., Farrokhi, A.: Intrusion detection using reduced-size rnn based on feature grouping. Neural Comput. Appl. 21(6), 1185–1190 (2012)
20. Moustafa, N., Slay, J.: The evaluation of network anomaly detection systems: statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. Inf. Secur. J. A Glob. Perspect. 25(1-3), 18–31 (2016)
21. Moustafa, N., Slay, J.: The unsw-nb15 data set description, Unsw. adfa. edu. au. [Online] https://www.unsw.adfa.edu.au/unsw-canberracyber/cybersecurity/ADFANB15-Datasets/ (2016). Accessed 10 May 2020
22. Moustafa, N., Slay, J.: Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6. IEEE (2015)
23. Hooshmand, M.K., Gad, I.: Feature selection approach using ensemble learning for network anomaly detection. CAAI Trans. Intell. Technol. 5(4), 283–293 (2020)
24. Ganganwar, V.: An overview of classification algorithms for imbalanced datasets. Int. J. Emerg. Technol. Adv. Eng. 2(4), 42–47 (2012)
25. Kotsiantis, S., et al.: Handling imbalanced datasets: a review. GESTS Int. Transac. Comput. Sci. Eng. 30(1), 25–36 (2006)
26. Ramentol, E., et al.: Smote-rsb*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory. Knowl. Inf. Syst. 33(2), 245–265 (2012)
27. Liu, A., Ghosh, J., Martin, C.E.: Generative oversampling for mining imbalanced datasets. DMIN, 66–72 (2007)
28. Chawla, N.V., et al.: Smote: synthetic minority over-sampling technique. J. Artif. Intell. Res. 16, 321–357 (2002)
29. Abdel-Hamid, O., et al.: Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4277–4280. IEEE (2012)
30. Lee, C.-Y., Gallagher, P.W., Tu, Z.: Generalizing pooling functions in convolutional neural networks: mixed, gated, and tree. In: Artificial Intelligence and Statistics, pp. 464–472. PMLR (2016)
31. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)
32. Wu, H., Gu, X.: Max-pooling dropout for regularization of convolutional neural networks. In: International Conference on Neural Information Processing, pp. 46–54. Springer (2015)
33. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830 (2011)
34. Team, T.T.D., et al.: Theano: a python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688 (2016)
35. Sriram, S., et al.: Dcnn-ids: deep convolutional neural network based intrusion detection system. In: International Conference on Computational Intelligence, Cyber Security, and Computational Models, pp. 85–92. Springer (2019)
36. Potluri, S., Ahmed, S., Diedrich, C.: Convolutional neural networks for multi-class intrusion detection system. In: International Conference on Mining Intelligence and Knowledge Exploration, pp. 225–238. Springer (2018)
37. Sriram, S., et al.: Network flow based IoT botnet attack detection using deep learning. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 189–194. IEEE (2020)
38. Vinayakumar, R., et al.: A visualized botnet detection system based deep learning for the Internet of Things networks of smart cities. IEEE Trans. Ind. Appl. 56(4), 4436–4456 (2020)
39. Ravi, V., et al.: Adversarial defense: dga-based botnets and dns homographs detection through integrated deep learning. IEEE Trans. Eng. Manag. (2021)
40. Hartl, A., et al.: Explainability and adversarial robustness for rnns. In: 2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService), pp. 148–156. IEEE (2020)

How to cite this article: Hooshmand, M.K., Hosahalli, D.: Network anomaly detection using deep learning techniques. CAAI Trans. Intell. Technol. 7(2), 228–243 (2022). https://doi.org/10.1049/cit2.12078