# Design of a Bottleneck Layered DNN Algorithm for Intrusion Detection System

**S. Kavitha[1], J. Manikandan[2]**

[1]Professor, Department of ECE, Hindusthan Institute of Technology, Coimbatore, India
[2]Professor, Department of Mechanical Engineering, Hindusthan College of Engineering and Technology, Coimbatore, India

**E-mail:** [1]kavithamani2003@gmail.com, [2]manikandan.hcet@gmail.com

**Abstract**

Deep learning algorithms are very effective in the application of classification and prediction over the traditional estimators. The proposed work employs a bottleneck layer algorithm on CICIDS-2017 dataset to prove its efficacy on the prediction of cyber-attacks. The performance of the bottleneck model architecture is incorporated with Artificial Neural Network (ANN) and Deep Neural Network (DNN) models and compared over the traditional ANN, DNN and Support Vector Machines (SVM) models. The experimental work reaches a maximum accuracy of 92.35% in the DNN and 90.98% in ANN algorithm respectively.

**Keywords:** Intrusion detection system, bottleneck layer, cyber security model, anomaly prediction

## 1. Introduction

The rate of data communication is increasing day by day and it increases the chances of interruption over the communication phase in different mediums. The main moto of an Intrusion Detection System (IDS) is to identify the abnormalities in the network medium and intimate it to the source operator.

In some cases the security algorithm incorporated in the work will respond to the attack based on the detected malfunction of the network. In general, the IDS are categorized based on their application wise implementation. A host-based IDS is a model employed in the hardware modules to monitor its activity on data movement inside the hardware. It helps the user to track the abnormal changes in the access of the hardware units. In most cases the host based IDS are not having permission to take decision on the identified intrusions [1, 2]. The network based IDS modules are designed to monitor the malicious movement on the entire

*DOI: https://doi.org/10.36548/jsws.2021.4.004*
*Received: 17.03.2022, received in revised form: 21.04.2022, accepted: 06.05.2022, published: 16.05.2022*

network while data sharing process. The identification of threats in this model is employed by analyzing the metadata of the contents and packets transmitted. Though, in most cases the network IDS model is not visible to the user from their end. Implementation of either host IDS or network IDS is not efficient for protecting a network and that requires combined technology for providing a complete protection [3, 4].

The IDS detection methods are classified into signature detection, anomaly detection and hybrid detection types based on their operating natures. The signature model assigns a signature to an intrusion when it is found for the first time. The signature IDS models are very efficient in detecting the known threats. The anomaly detection method monitors the network behavior continuously and it will alert the user when it founds an abnormal activity in the packet data movement of the network. The hybrid IDS is a model combined with signature IDS and anomaly IDS which reduces the error rate significantly than the other methods [5, 6].

## 2. Literature Work

A feed forward deep neural network algorithm was employed for wireless IDS application through wrapper based feature extraction process [7]. An extra tree algorithm is utilized in the wrapper feature extraction model and that reaches up to 99.66% of accuracy on binary class intrusion prediction 99.77% on multiclass estimations. A hybrid algorithm based on deep network model and machine learning algorithm was designed to classify the intrusion available in UNB ISCX 2012 dataset [8]. A stacked autoencoder method was incorporated in the work to observe the latent feature from the training samples. Further the extracted features are moved into a SVM classifier for the analysis. The experimental result indicates an accuracy rate of 99.49% with computational time of 6.15 minutes.

An optimized CNN algorithm was incorporated with hierarchal multi-scale LSTM for improving the classification accuracy with the IDS model [9]. The hyper parameters extracted from the feature set of the dataset are analyzed with lion swarm optimization for tuning purposes. The experimental projection indicates an accuracy range of 90% on the intrusion detection process. A network intrusion detection algorithm was designed using deep autoencoder module for feature extraction and deep neural network algorithm for classification. The classification accuracy of the proposed model is increased by having a hyperparameter optimization procedure [10]. The experimental projection indicates an accuracy attainment of 83.33% while using NSL-KDD dataset.

A hybrid model was structured using the traditional CNN and weight dropped LSTM model for estimating the intrusions on big data applications [11]. A 97.17% of accuracy was achieved in the model with UNSW-NB15 dataset. A conditional random field and linear correlation coefficient-based feature selection model was developed to utilize the most optimum features for the classification process using CNN model [12].

An experimental was conducted in the work with KDD dataset and that reaches to 98.88% of accuracy in the estimation process. A bi-directional LSTM model was proposed to classify the U2R and R2L intrusions [13]. The presented experimental study shows an accuracy of 91.36% on its maximum. A RNN based intrusion detection algorithm was developed to identify the malicious activity from the network data and an improvement of 16.67% was detected over the previous techniques.

## 3. Dataset Description

### 3.1 CICIDS-2017 dataset

The real world network data is collected in the CICIDS dataset through a CIC flow meter. The dataset includes 79 labels of information along with 80 features of data. The real time data is extracted by monitoring the network activity of 25 different location users at the same time for 5 days. The data collected in the dataset are categorized through a McAfee report as DOS, DDOS, brute force SSH, FTP, botnet, infiltration and web [14]. Table 1 indicates the list of information available on all the five days.

**Table 1.** Details Available in CICIDS Dataset

| Class Type | Days |
|---|---|
| Benign | Monday to Friday |
| FTP & SSH Patator | Tuesday |
| DoS: Goldeneye, Hulk, lowhttptest, slowlories, Heartbleed | Wednesday |
| XSS, Brute Force, Injection, SQL, Infiltration | Thursday |
| DDOS, BOT, PortScan | Friday |

### 3.2 Data imbalance processing – CICIDS2017 dataset

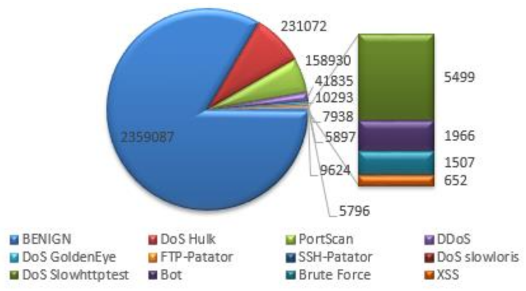

**Figure 1.** Methodology of the proposed work

The CICIDS 2017 dataset is a highly imbalanced dataset, an upsampling and downsampling technique was implemented over it to normalize the available data. There are 12 class of information available in the dataset that includes benign, DoS Hulk, PortScan, DDoS, DoS GoldenEye, FTP-Patator, SSH-Patator, DoS SlowIoris, DoS Slowhttptest, Bot, XSS and Brute force attacks. Figure 1 represents the class distribution of each instances and it clearly indicates that the balancing is not possible with such huge count difference data samples. Therefore the work selected only the top four classes of data in the available sample count. Table 2 indicates the actual and balanced data information after the performed sampling process.

**Table 2.** External Server Class Distribution

| Selected Instances | Actual | Balanced |
|---|---|---|
| BENIGN | 2359087 | 1035907 |
| DoS Hulk | 231072 | 142814 |
| PortScan | 158930 | 129745 |
| DDoS | 41835 | 125389 |

## 4. Proposed Method

The workflow followed in the proposed work is shown in Figure 2. The balanced data mentioned in Table 2 is forwarded for data normalization process. The data with abnormality and missing values are removed or edited in the step. The normalized data are further categorized as training and testing data at 70:30 ratio. A customized feature tuning operation is employed next to the normalization process for selecting the optimum features for the estimations. Based on the selected feature, a neural network algorithm is trained in the work for its classification process. In the proposed work a bottleneck layered ANN and DNN model are designed to analyze its efficiency.

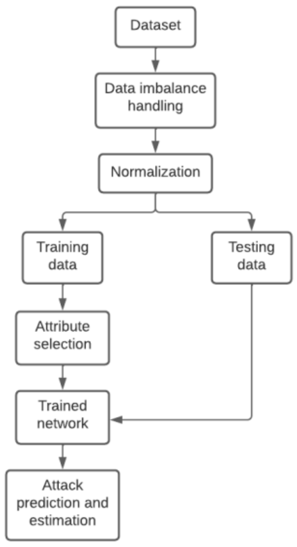

**Figure 2.** Proposed workflow

## 4.1 Bottleneck layer

The training process of the bottle neck layer networks are performed with the following objective function.

$$E = \frac{1}{x} \sum_{i=1}^{x} \left\| e_i^2 \right\| \tag{1}$$

where, $e_i$ = reconstruction error = $n_1 + \hat{n}_i$

The range between the $n_1$ and $\hat{n}_i$ represents the count of hidden neurons included between the input and output layer. In the proposed work the total count of hidden layer included between the input and output layer is three counts including the bottle neck region. The first and last hidden layers are employed in the work for encoding and decoding process. $e_n$ indicates the encoding layer with the activation function as $a_1$ from the input layer to the first encoding layer $n$.

$$e_n = a_1((v^x\,(X) + b^x)_k) \tag{2}$$

where,

$v^x$ denotes the matrix weightage of m x n,

$b^x$ represents the vector length of the column m and k= replicates the differences from 1 to m

A nonlinear principal component of $y$ is mapping $a_1$as their second order function for the activation at bottleneck layer.

$$y = a_2(v^x (e_n) + b^{-x}) \tag{3}$$

The decoding layer activation function is represented as $d$ with the activation function of $a_2$ from the bottleneck layer output.

$$d = a_3((v^p (y) + b^p)_k) \tag{4}$$

The final and the output function is extracted from the decoded layer $d$ with the vector values as $e_i$

$$e_i = a_4((v^p (d) + b^{-p})_i) \tag{5}$$

Therefore, the objective function $E$ is reduced by analyzing the optimal values from $v^x$ to $b^{-p}$. Similarly, the hyperbolic transfer function is implemented for $a_1$ and $a_2$, at the same time $a_3$ and $a_4$ are operated with an identity function. Hence the principle component $y$ and vector value $e_i$ is formatted as,

$$y = (v^x (e_n) + b^{-x}) \tag{6}$$

$$e_i = (v^p (d) + b^{-p})_i \tag{7}$$

## 4.2 Attack classification in bottleneck layer

The residual features extracted from the training samples are considered for the attack classification by the bottleneck layer network. It is done by estimating the squared prediction error $S_{err}$ from the fitness lagging ratio from the layered network. The $S_{err}$ detection index is calculated as follows for a timeframe $T$.

$$S_{err}(T) = e^K(T)e(T) = \sum_{i=1}^{y} e_i^2 (T) \tag{8}$$

The approximation of $S_{err}$static distribution is generated as,

$$S_{err} \infty gx_h^2 \tag{9}$$

where, mean moments of matching ‘m’ and variance ‘v’ are incorporated in the prediction of freedom degree ‘h’ along with the weight ‘g’.

$$g = \frac{v}{2m} \tag{10}$$

$$h = \frac{2m^2}{v} \tag{11}$$

Therefore, the maximum control range is estimated as

$$U_{lim} = (g)\left(x_{1-\partial}^{2}\ \right)(h) \tag{12}$$

where, $\partial$ indicates the predefined range of significance.

The confidence range of $S_{err}$ represents the achieved prediction level from the previous analysis as $U_{lim}^{2}$.

The prediction accuracy of the bottleneck layer algorithms can be improved by adding a moving average filter. Therefore the false alarm rate caused by the noises from the training samples are avoided. A general moving average function employed in the bottleneck layer algorithm is represented below.

$$e(k) = (1-\omega)\, e(k-1) + \omega e(k) \tag{13}$$

$$U_{lim}(k) = \ \|e(k)\|^2 \tag{14}$$

where, $\omega$ = diagonal elements of the matrix from the residual forgetting factors.

## 4.3 Artificial Neural Network (ANN)

The artificial neural network model is a single hidden layer network where there is no encoding or decoding layer available separately in the workflow. The mathematical model of a feed forward neural network is presented in Figure 3.

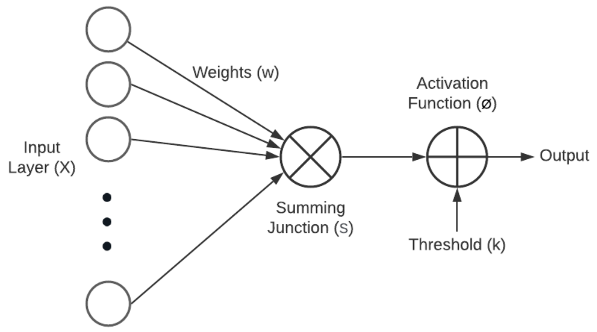

**Figure 3.** Mathematical model of ANN

The ANN models can also segregated with respect to the information movement between the input and output layer. Therefore in most of the neural networks, the data are analogs in generating the activation function. There are some neural networks that generates a discrete function for the input values in binary forms. In such cases an echelon function is created for activating the neurons.

Assume that the input layer of the ANN is forced with various input features (x) extracted from the given input data. A summing function (S) is included next to the input layer for adding some values (w) to the extracted features. This improves the computational efficiency of the neural network to certain extent. The obtained summing outputs are averaged with a bias (b) value for producing a net income to the hidden neuron.

$$S = \sum_{i=1}^{n} x_1 w_1 + b \tag{15}$$

An activation function (∅) is added next to the summing function before creating an output to the further neurons.

$$Output = \emptyset(\sum_{i=1}^{n} x_1 w_1 + b) \tag{16}$$

The threshold (k) of the activation function can be operated with a learning rule at the time of the training process. The outcome of the activation function is same as of its inputs operated with a constant value (c).

$$\emptyset = cs \tag{17}$$

The value of c is calculated with the following function when it is a derivative.

$$c = \frac{dg}{ds} \tag{18}$$

where, g represents the argument differentiation of summing function (s).

A nonlinear activation function generates a continuous output between 0 to 1 and it is represented as follows.

$$\emptyset = \frac{1}{1+ e^{-s}} \tag{19}$$

The step function is employed in the nonlinear activation with a sigmoid model that allows to skip the values that are coming again and again for formatting a smooth output. Therefore the activation function becomes,

$$\emptyset_d = \frac{e^{-s}}{(1+ e^{-s})^2} \tag{20}$$

A hyperbolic tangent function given here when the output values are need to be observed from -1 to 1 in a continuous motion with in a nonlinear activation function and that is achieved with the sigmoid operation.

$$\emptyset = \frac{e^s - e^{-s}}{e^s + e^{-s}} \tag{21}$$

The derivative equation of the hyperbolic tangent function is derived as follows.

$$\emptyset_d = \frac{(e^s + e^{-s})^2 - (e^s - e^{-s})^2}{(e^s + e^{-s})^2} \tag{22}$$

ReLU is yet a major function applied widely in the ANN and that projects the output same as like of the linear function.

$$\emptyset = \max(0, s) \tag{23}$$

A minor value of constant is applied to the ReLU function when the function is need to be started from -1 instead of 0 and such function is termed as Leaky ReLU which is formulated as below.

$$\emptyset = \max(cs, s) \tag{24}$$

## 4.4 Deep Neural Network (DNN)

Training a single layer neural network requires a huge computational time and memory space. The computational complexity may go beyond the acceptable range when the hidden layer networks are increased in it. In order to manage such complexities a cascade formation of neural network was proposed [15].

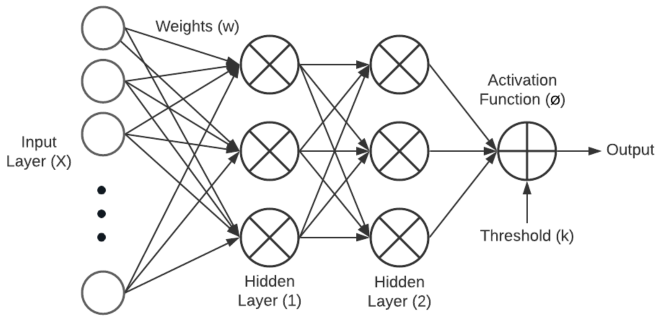

**Figure 4.** Architectural overview of DNN

The model makes the nodes of the present hidden layer into freeze mode for some time to enable the training process for the remaining nodes. Therese kind of freeze steps are usual in the traditional back propagation networks for reducing its complexity. However, in the developed DNN model the same is introduced for the two hidden layer feed forward neural network. The same model can be used for making ANN with any number of hidden layers. Figure 4 indicates the architectural view of the DNN model.

The DNN design follows the multilayer perception model in the transformation of feed forward signals. Similarly, the node connections between the hidden and output layers are having the ability to operate with any kind of transfer function. The hidden layer (1) presented in the model receives the input signals taken from the given dataset but the hidden layer (2) receives the weighted average input from the given input data along with the output of the 1st hidden layer. This makes the data presence in hidden layer (2) as more informative than the previous hidden network. In the same way the output layer receives the summation output of both 1st and 2nd hidden layers.

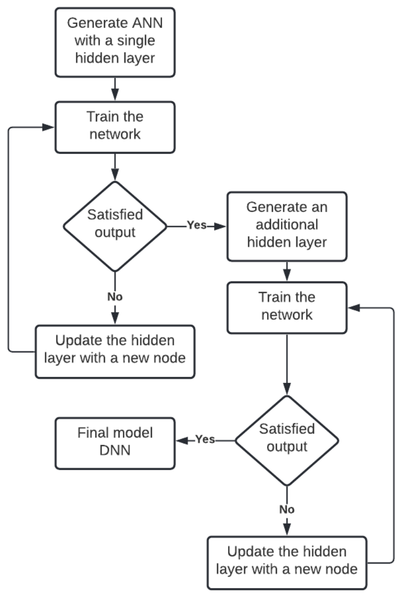
Generate ANN with a single hidden layer
Train the network
Satisfied output
Yes
No
Generate an additional hidden layer
Update the hidden layer with a new node
Train the network
Satisfied output
Final model DNN
Yes
No
Update the hidden layer with a new node

**Figure 5.** Formation of DNN from ANN

The formation of DNN is shown in Figure 5, where the initial process starts from making a three layer ANN with input layer, hidden layer and output layer. The node connection at the input and output layers are determined with respect to the problem that is going to be handled in the respective work. However, the hidden layers are equipped with only one node in the beginning and random weights are assigned to the nodes till observing an error (E) that not minimizing the iterations.

$$E = \frac{op_{max} - op_{min}}{NS} \sum_{N=1}^{N} \sum_{i=1}^{S} (Xs - Ys)^2 \tag{25}$$

where,

$op_{max}$ and $op_{min}$ are indicating the output range of the considered problem.

N = Number of nodes in the output,

S = Count of the samples in the training data,

Xs and Ys are representing the generated values of each node.

The error calculation finalizes the total number of node connection need to be included in the hidden layer 1. After achieving a satisfied outcome from the hidden layer 1, the formation of hidden layer 2 is generated in the model. The weightage W of each node is processed with respect to the following equation on the training process.

$$W(n+1) = \frac{\left(Xs\ (n) + Ys\ (N)\right)W(n)}{Xs + Ys\ (n+1)} \tag{26}$$

Node addition is the process of cell division that splits the node which is available in the hidden layer of the ANN. The newly formatted node also contains the same neuron connections placed in the previous nodes and their weights are calculated as follows,

$$W_1 = (1+\alpha)W \tag{27}$$

$$W_2 = (-\alpha)W \tag{28}$$

where,

W = existing node weight vector,

$W_1$and $W_2$ are the weight vectors of the newly generated nodes

α = Random or fixed mutation value assigned to the nodes

Therefore the newly generated node creates the weight vector without a random generated value and it makes the developed neural network model as less complex model.

## 4.5 Multiclass Support Vector Machines (MSVM)

SVM classifiers are basically formatted to operate with binary classification. Therefore the same binary process is operated on MSVM by doing various binary calculations and it is represents as one vs one model of SVM. In general the classification starts by making class A as X and remaining all to a same group Y. The total number of classifiers required for the SVM is analyzed with the following term and the data points model of MSVM is shown in Figure 6.

$$\frac{n(n-1)}{2} \tag{29}$$

where, n = total number of classifiers required for the one vs one classification.

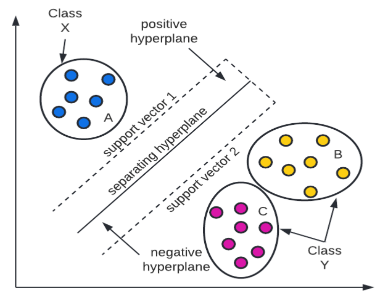

**Figure 6.** Datapoints of MSVM

The SVM classifiers are formatted with a hyperplane to make a decision margin among the two class samples. The surfaces of the hyperplanes can be increased or increased for tuning the accuracy of the prediction model. The SVM classifiers are generally good in classifying the problems with pattern classification. Apart from hyperlane surface, support vectors are there for each class. It is helpful for classification of numerical or alphabetic data. The support vector values are very close to the hyperplane and placed over the boundaries on segregating the classes [16].

The following are the two operations implemented in building the SVM mathematically,

- The input vectors are considered as a nonlinear parameter in a high dimensional feature space.
- Generation of a hyperplane with respect to the extracted features.

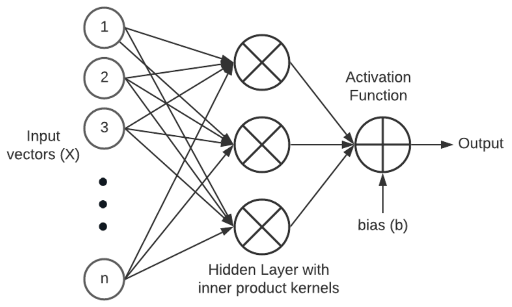

**Figure 7.** Architecture of SVM

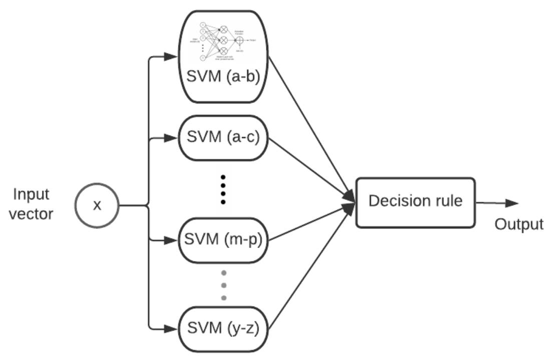

**Figure 8.** Architecture of MSVM

An architectural overview of SVM and MSVM is shown in Figure 7 and 8, where in both cases the input signals are represented as a vector x. However, in MSVM 'n' number of SVM are placed according to the classes of data to be identified from the given dataset. Therefore the hidden layers of each SVM has an operated product kernel based on the input and output vectors. Hence the hidden layer gives a linear output and those are grouped in the output neuron and that is computed as,

$$Output = \sum w_i k\ (x, s_i) + b \tag{30}$$

where,

x = input vector,

$s_i$= output vector,

$k$ = kernel of $(x, s_i)$

b = bias and $w_i$= weight vector.

The weight and bias values are estimated in the training process with respect to the output prediction accuracy. The product of kernel is employed in the work to observed linear output from the predictions.

## 5. Experimental Work

### 5.1 Performance of the External server data – CICIDS2017 dataset

Table 3 and 4 indicate the performances of the bottleneck layer algorithms over the regular approaches. Table 3 represents the performances of the actual data and Table 4 projects the attainments of the balanced data.

**Table 3.** Performance of Actual Data

| Algorithm | Accuracy | TPR | Precision | f1 score |
|---|---|---|---|---|
| ANN | 79.35 | 78.65 | 78.12 | 79.3 |
| DNN | 80.51 | 80.41 | 79.65 | 80.35 |
| SVM | 78.15 | 78.54 | 79.21 | 78.19 |
| ANN Bottleneck | 83.21 | 82.47 | 82.79 | 82.5 |
| DNN Bottleneck | 84.89 | 84.21 | 83.34 | 84.6 |

**Table 4.** Performance of Balanced Data

| Algorithm | Accuracy | TPR | Precision | f1 score |
|---|---|---|---|---|
| ANN | 83.14 | 82.13 | 83.4 | 82.89 |
| DNN | 84.7 | 84.42 | 83.74 | 84.57 |
| SVM | 82.46 | 81.24 | 82.78 | 80.98 |
| ANN Bottleneck | 90.98 | 90.35 | 91.05 | 89.78 |
| DNN Bottleneck | 92.35 | 91.84 | 93.22 | 90.1 |

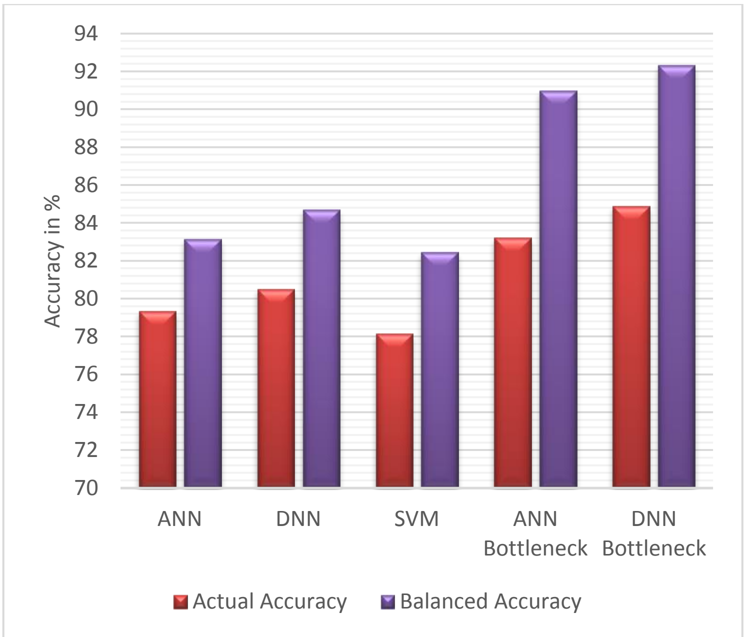

**Figure 9.** Accuracy comparison of actual and balanced CICIDS2017 data

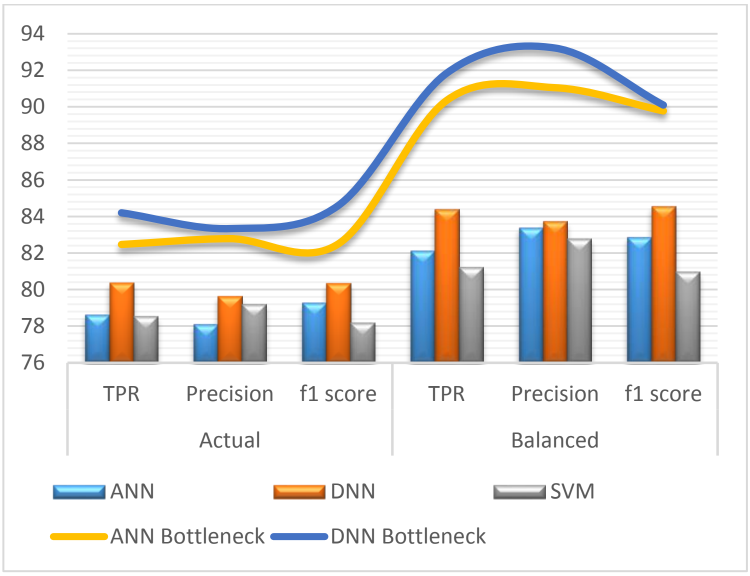

**Figure 10.** Comparison of actual and balanced data on CICIDS2017 dataset

The accuracy differences among the actual and balance CICIDS2017 data shows a huge variation. Especially the differences are massive while the network is trained with a bottleneck layer. Figure 9 explores the accuracy differences between models. Figure 10 explores the performances of the CICIDS2017 dataset in terms of TPR, precision and f1score. It shows that the performances of the DNN bottleneck are better than the other algorithms.

## 6. Conclusion

ANN and DNN are the primary deep learning algorithms widely used for IDS estimations. However, a fine tuned algorithm will improve the efficiency of the prediction algorithm. Therefore, a bottleneck layered architecture is employed in the proposed work with the traditional ANN and DNN algorithms and its performances are compared over the traditional ANN, DNN and multiclass SVM. The experimental work conducted with CICIDS2017 dataset shows a major accuracy difference on the bottleneck layered algorithms over the traditional ones. Especially the DNN model structured with bottleneck layer improves the prediction accuracy in a better rate.

## References

[1] Smys, S., Abul Basar, and Haoxiang Wang. "Hybrid intrusion detection system for internet of things (IoT)." Journal of ISMAC 2, no. 04 (2020): 190-199.

[2] Gupta, Akshay Ramesh Bhai, and Jitendra Agrawal. "Machine Learning-Based Intrusion Detection System with Recursive Feature Elimination." In Inventive Computation and Information Technologies, pp. 157-172. Springer, Singapore, 2021.

[3] Sathesh, A. "Enhanced soft computing approaches for intrusion detection schemes in social media networks." Journal of Soft Computing Paradigm (JSCP) 1, no. 02 (2019): 69-79.

[4] Solani, Sona, and Nilesh Kumar Jadav. "A survey on intrusion detection system using artificial intelligence." In International conference on Computer Networks, Big data and IoT, pp. 67-80. Springer, Cham, 2019.

[5] Chen, Joy Iong Zong. "Smart security system for suspicious activity detection in volatile areas." Journal of Information Technology 2, no. 01 (2020): 64-72.

[6] Sridevi, R., and N. Nithya. "Intrusion Detection System Using WoSAD Method." In Inventive Communication and Computational Technologies, pp. 423-430. Springer, Singapore, 2020.

[7] Kasongo, Sydney Mambwe, and Yanxia Sun. "A deep learning method with wrapper based feature extraction for wireless intrusion detection system." Computers & Security 92 (2020): 101752.

[8] Mighan, Soosan Naderi, and Mohsen Kahani. "A novel scalable intrusion detection system based on deep learning." International Journal of Information Security 20, no. 3 (2021): 387-403.

[9] Kanna, P. Rajesh, and P. Santhi. "Unified deep learning approach for efficient intrusion detection system using integrated spatial–temporal features." Knowledge-Based Systems 226 (2021): 107132.

[10] Kunang, Yesi Novaria, Siti Nurmaini, Deris Stiawan, and Bhakti Yudho Suprapto. "Attack classification of an intrusion detection system using deep learning and hyperparameter optimization." Journal of Information Security and Applications 58 (2021): 102804.

[11] Hassan, Mohammad Mehedi, Abdu Gumaei, Ahmed Alsanad, Majed Alrubaian, and Giancarlo Fortino. "A hybrid deep learning model for efficient intrusion detection in big data environment." Information Sciences 513 (2020): 386-396.

[12] Riyaz, B., and Sannasi Ganapathy. "A deep learning approach for effective intrusion detection in wireless networks using CNN." Soft Computing 24, no. 22 (2020): 17265-17278.

[13] Sohi, Soroush M., Jean-Pierre Seifert, and Fatemeh Ganji. "RNNIDS: Enhancing network intrusion detection systems through deep learning." Computers & Security 102 (2021): 102151.

[14] Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." ICISSp 1 (2018): 108-116.

[15] Islam, Md Monirul, and Kazuyuki Murase. "A new algorithm to design compact two-hidden-layer artificial neural networks." Neural Networks 14, no. 9 (2001): 1265-1278.

[16] Frimpong, Emmanuel Asuming, Philip Yaw Okyere, and Johnson Asumadu. "Prediction of transient stability status using Walsh-Hadamard transform and support vector machine." In 2017 IEEE PES PowerAfrica, pp. 301-306. IEEE, 2017.

## Author's biography

**S. Kavitha** works as a Professor in the Department of ECE at Hindusthan Institute of Technology, Coimbatore, India. Her research are includes WSN, mobile networks, optimization techniques, automation systems, artificial intelligence, computer vision, network security and cloud computing.

**J. Manikandan** is currently a Professor in the Department of Mechanical Engineering, Hindusthan College of Engineering and Technology, Coimbatore, Tamil Nadu, India. His research interest includes deep learning algorithms, automation systems, process control, computational fluid dynamics, 3D analysis and material sciences.