International Journal on Advanced Science Engineering Information Technology

Vol.13 (2023) No. 2
ISSN: 2088-5334

Botnet Detection Model in Encrypted Traffics Software-Defined Network (SDN) Using Deep Neural Network (DNN)

Rio Suneth [a], Heru Sukoco [a,*], Shelvie Nidya Neyman [a]

[a] *Department of Computer Science, Faculty of Mathematics and Natural Sciences (MIPA), Bogor Agricultural University (IPB), Bogor, 16680, West Java, Indonesia*

Corresponding author: [*]hsrkom@apps.ipb.ac.id

***Abstract*— The presence of network technology eliminates regional boundaries that become obstacles in communicating and exchanging data and information to the public. The wider the zone of a network, the network infrastructure will increase in size. The bigger the network infrastructure, the higher the level of management complexity. The Software Defined Network (SDN) concept is a new network concept that provides a solution for managing large infrastructure networks and has a wide service zone. SDN architecture is different from traditional networks. The SDN architecture is divided into three: the data plane, control plane, and application plane. Whereas in the traditional network architecture, the three are combined into one. Besides, in maintaining network security. SDN offers a security system, namely the OpenFlow Protocol. The OpenFlow Protocol security system works to regulate the packet traffic that passes. Forwards registered packet data traffic and performs down the action for unknown packet traffic. The weakness is that the OpenFlow Protocol must always be updated with SDN network packet traffic, and the system cannot detect the threat of attacks on encryption traffic. Nowadays, the frequency of attacks on network traffic is relatively high. The attack techniques used also evolved. The techniques used are also evolving. Botnets have been able to use several encryption protocols such as TLS / HTTPS, Tor, and P2P as loopholes to attack a network. SDN's presence as a management solution for large infrastructure networks is not directly proportional to its security system that undoubtedly have a bad impact on SDN network users. Therefore, this study aims to develop an SDN Network Intrusion Detection System (IDS) model to detect botnets in encryption traffic. The model was developed using the Deep Neural Network (DNN) approach. The SDN network botnet detection model developed can detect encryption traffic botnets with an accuracy rate of 94.78%, 93.28% precision, and a recall of 99.11%.**

***Keywords*— Botnet; deep neural network; encrypted traffic; software-defined network.**

Manuscript received 25 Jul. 2019; revised 12 Apr. 2022; accepted 28 Feb. 2023. Date of publication 30 Apr. 2023.

I. INTRODUCTION

Approximately 16-25% of computer devices connected to public networks are infected with botnets. Therefore, in securing information traffic. Network administrators utilize the encryption protocol, however, in carrying out attacks on the network. The botnet uses several encryption protocols such as TLS / HTTPS, Tor, and P2P as loopholes to attack and steal information in network traffic [1]. To overcome botnet attacks in network traffic, administrators take advantage of the Software-Define Network (SDN) network concept.

SDN is a new paradigm in the world of networking. The basic concept of SDN networks in managing networks is by separating control plane devices and data planes. The SDN structure is divided into three main layers: application, control plane, and data plane. Plane and application controls are associated with the Application Programming Interface (API). Data plane and control plane are connected to the OpenFlow protocol.

OpenFlow is a necessary forwarding switch device that forwards packets according to the flow table rules. Flow tables contain a set of rules consisting of matching fields, counters, and instructions. Matching header fields (Ethernet, IPV4, IPV6, or MPLS). Calculation by collecting flow statistics for the number of packets, bytes, and duration of flow. The instruction is an action command against the package. OpenFlow protocol has many versions, and each version has a difference in the set of flow table rules. OpenFlow version 1.0 is the most used, and in October 2013 version 1.4 was released as the latest version. For improving the SDN protocol, OpenFlow security system must always be updated. If the OpenFlow version used is an old version, and

the controller is attacked, the attacker will quickly master the SDN network infrastructure [2].

Previous research was conducted by Wijesinghe, Tupakula, and Varadharajan [3] on OpenFlow protocol version 1.4 for detecting botnets on SDN networks. Furthermore, Chi *et al* [4] studied the integration of intrusion detection system (IDS) in SDN for the security of the advanced metering infrastructure (AMI) system. In this case, Li, Meng, and Kwok [5] proposed specifications for OpenFlow. According to him, OpenFlow missed the TLS protocol's security, which is often used as a botnet loophole to carry out attacks such as spoofing, piracy, DoS, and several other attacks. Neu *et al* [6] developed a new IDS instead of traditional IDS and firewall detected attacks on encrypted data. Research by Jing *et al* [7] develops detection of real-time botnets called bot-guards. The method used is graph theory by giving the lens imaging graph (CLI-graph) convex to the controller to describe the botnet's characteristics. The result is a high accuracy of 90%.

According to Hadianto and Purboyo [8], SDN as a new concept that offers ease of centralized SDN security management, needs to be improved. As a programmable network, SDN can easily accept innovative ideas, especially the improvement in the security system it has. Therefore, this study aims to build a botnet detection model on traffic encryption for SDN network traffic. The research's scope is to recognize the pattern of botnet attacks on encryption traffic in SDN network traffic collected from secondary data CTU-malware-capture-botnet datasets (CTU-13). Then build a botnet detection model using the deep neural network (DNN) approach.

II. Material and Method

A. *Software-Define Network* (SDN)

According to Su *et al.* [9] and Rana, Dhondiyal, and Chamoli [10], conceptually, SDN network architecture is divided into three layers: application plane, control plane, and data plane. Between the application plane and the control, the plane is connected to the application program interface (API). In the control plane layer, there are many and various network topology mappings. The control plane maps topology on network devices and simultaneously regulates packet traffic routing in the network. SDN access control plane using application web dashboard API [6]. The control plane is a very programmable device because the control plane can use any programming without being limited by specific vendors. This layer is a collection of network routing devices such as routers and switches for a data plane. Between the data plane and the control, the plane is connected to the OpenFlow protocol [11].

B. *Botnet*

The botnet is a massive and organized collection of computers (zombies). The botnet is controlled by a botmaster or herder, both an attacker and a botnet manager [10]. The purpose of a botnet made by an attacker is to obtain financial benefits. Among various forms of malware, botnets are the most severe and dangerous attacks. It is a serious and dangerous attack because botnets can carry out attacks without being detected. Where botnets use encrypted protocols to hide in carrying out attacks and malicious activities such as sending spam e-mails, phishing, click fraud, DDos, and spreading other malicious software [12], [13], [14].

C. *Encrypted Traffic*

Encrypted traffic is a protocol that provides confidentiality or better known as the encryption protocol. Encryption protocols usually offer communication authentication, data integrity, data protection, and none-repudiation. In general, all encryption protocols' working principle is the same and is divided into several phases, namely, in the first phase, connection initialization and encrypted data transportation are carried out. The first phase can still be divided into several phases, namely initial handshake, authentication, and creating a secret key. The secret key is used to encrypt the data transferred in the second phase. Protocols that are classified as encryption protocols include internet protocol security (IPsec), transport layer security (TLS), secure shell protocol (SSH), BittToren HTTPS, SSL and Skype [9].

D. *Deep Learning*

Deep learning is a complicated version of machine learning. It is said to be complicated because it has a level of abstraction in every hidden layer. Deep learning is one method that is considered appropriate enough to be used to read the structure of complex datasets. Deep learning is often known as hierarchical learning or deep structural learning in order to read the structure of complex datasets. Deep learning architecture that is commonly used is the sincere belief network (DBN), deep neural network (DNN), and recurrent neural networks (RNN). The architecture has been widely used in the field of research, namely: speech recognition, computer vision, audio recognition, machine translation, and social network filtering [15].

According to Dong, Wang, and He [16], the deep learning method's core is its ability to automatically extract features in many inner layers. Deep learning algorithm provides excellent performance in solving complex problems. However, even though the deep learning algorithm for getting excellent performance requires configuration and accurate parameter determination. Without the right selection of parameters, a model built using the deep learning algorithm produced poor performance.

E. *Deep Neural Network* (DNN)

Deep neural network (DNN) is one of the classification methods found in deep learning [17]. DNN is an artificial neural network that has more than one layer of hidden layer units. Some hidden layer layers can solve the problem of calcification with complex data. Each hidden layer can learn features at different levels of abstraction. [18], in addition to being able to overcome the limitations of the hidden layer Neural Network (NN), the core of a deep neural network (DNN) can also extract features through the hidden layer automatically. In general, DNN architecture is the same as the architecture of the Neural Network (NN), which has three main layers: input, hidden, and output layer. The input layer includes the number of input parameters. The hidden layer is the number of neuron sets used. The output layer is the output of the process carried out by the hidden layer. The number of neurons and the number of hidden layers in the DNN model architecture do not have a maximum limit, which

distinguishes DNN from NN. The depth of the DNN architecture is seen from the number of hidden layers used and the type of hidden layers and linear or non-linear functions [19].

F. Methods

The development of the detection model focuses on botnet attack patterns on encrypted traffic on the SDN network. The development process is divided into several different stages: data acquisition, Deep neural network consisting of Data Model Transformation, Data Distribution model, development of model architecture, model training, model testing, and model evaluation. Several stages in the research are shown using a flow diagram in Figure 1.

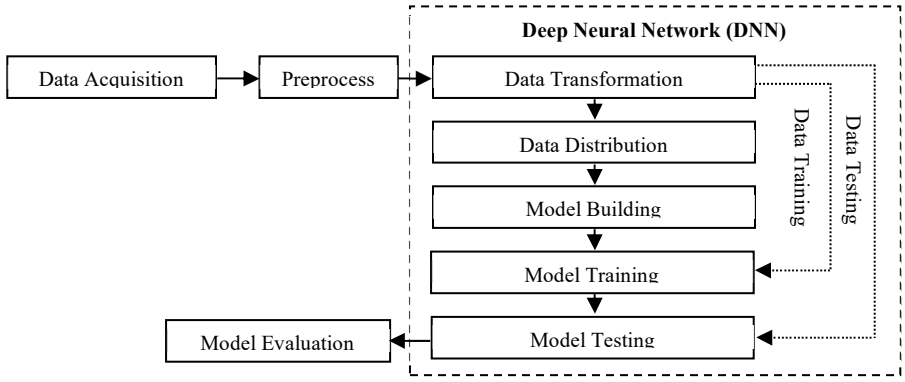

Fig. 1 Research methods

G. Data Acquisition

The process of acquiring network traffic uses secondary data, namely CTU-malware-capture-botnet datasets. Capture Lab results. Czech Network Technical University (CTU) in 2011. The data is published publicly and can be downloaded at https://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal -and-background-traffic.html. The CTU-malware-capture-botnet dataset is a collection of traffic data with 13 capture scenarios of PCAP file types. The acquisition of botnet traffic data and regular is only made on 10 PCAP capture scenarios. The acquisition process is carried out using the Wireshark application's help and the information provided on the download address. The results of the acquisition of botnet traffic data and standard ten scenarios are converted to CSV files. The amount of traffic data obtained is 52,701 records and 49 features.

H. Preprocess

The preprocess stages carried out are data cleaning, labelling, and feature selection. Data cleaning is the stage of cleaning data from a missing value (NA), noisy or outlier, and data inconsistencies. The number of traffic obtained in the cleaning data process is 28,681 records and 17 new features from 49 features. Labeling is the process of grouping data into several class targets or categories [20]. Labeling is needed for the classification process of supervised learning to label the target class as output data. There are two target classes as output that were sent, namely botnet and normal. Next, the next step is to select 17 new features obtained during the data cleaning process. Feature selection aims to get the best features and influence in reflecting the target class datasets. Therefore, feature selection is essential, and in this case, feature selection is carried out based on the info gain value on the decision tree algorithm defined in equation (1).

$$\text{infogain(class,attribute)} = H(class) - H(class|attribute) \quad (1)$$

Where H represents the entropy of class and attributes defined in equation (1), before calculating the info gain of the 17 new features, the calculation is first made of the entropy value of the 17 features, where H represents the entropy of class and attribute defined in equation (2) [7].

$$H(Y) = -\sum_{j=i}^{n} P(y_i).log\ 2\ .\left(P(y_i)\right) \quad (2)$$

Attribute or feature entropy probability of representing Y's appearance in the sample (Qian and Qiu 2014). The info gain of each feature in Table 1 illustrates the influence of a feature on the target class.

TABLE I
INFO GAIN NEW ATTRIBUTE

No	Attribute	Info Gain
1	Frnum	0.089242981
2	Frlength	0.529487080
3	Frtimerelative	0.121711681
4	Frtimedelta	0.345625732
5	Timesincefirstfr	0.274410177
6	Ipsrc	0.637844970
7	Ipdst	0.777239369
8	Ipid	0.054491906
9	Ipchecksum	0.024829787
10	ip_len	0.555283767
11	ip_flags	0.007087164
12	Srcport	0.767573656
13	Dstport	0.772748199
14	Checksum	0.000000000
15	Tcpflags	0.025942096
16	Tcpanalysispushbytessent	0.567903101
17	Windowsize	0.727264948

Table 1 shows how good and high the influence of 17 features on the target class. Then a selection of 17 features based on info gain values was carried out. Selection is made by cracking. Infogain 17 features were cracked with a limit value of ≥ 0.1. The value of 0.1 is the optimal value for cracking. Features with info gain ≥ 0.1 were selected as the best feature in Figure 2.

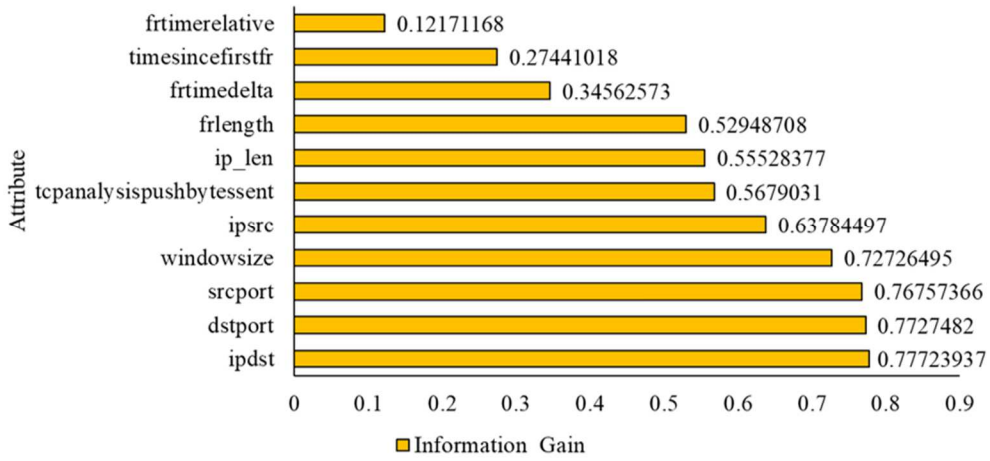

Fig. 2 Feature info gain cracking results

In Figure 2. The ipdst feature with the highest value 0.777239369 is the highest info gain feature placed as the root node feature. Furthermore, the root node feature is followed by ten other best features: *dstport, srcport, windowsize, ipsrc, tcpanalysispushbytesent, ip_len, frlength, frtimedelta, timesincefirst, and frtimereltive*.

III. Results and Discussion

A. Data Model

TABLE II
DATASETS OF THE BOTNET AND NORMAL TRAFFIC

Scenario	*Class Botnet*	*Class* Normal	Total
1	613	738	1351
2	582	1431	2013
3	400	619	1019
4	33	38	71
5	94	144	238
6	12	13	25
7	19	48	67
8	1852	2547	4399
9	838	894	1732
10	5589	12177	17766
Total	10032	18649	28681

The data model is traffic data that is used in the deep neural network (DNN) model. The data has 11 input and target class parameters as outputs. The model data used amounts to 28,681 traffic records consisting of 10,032 botnet classes and 18,649 standard record classes. The number of the botnet and normal class data from each scenario is shown in Table 2.

B. Model building environment

The DNN model is built using the R-Studio application with a robust framework. Hard is one of the quite popular frameworks from several other DNN frameworks such as Thano, Tensorflow, and H2O. Hard itself is a high-level neural network Application Programming Interface (API), which was developed with a focus on activating fast experiments [13]. The DNN model is built on the Asus A442U mobile PC, with Intel Core i5-7200U CPU specifications up to 3.1GHz, 4GB memory, and 1TB HDD.

C. Model data transformation

Before the data is used, the data was transformed first. The transformation process carried out is two, namely normalization, and matrix byte factor transformation.

1) Normalization: Normalization is done because the data distribution is very diverse, so it is necessary to do data uniformity. The normalization process is carried out on 11 new features that have been selected using the min-max method defined in equation (3).

$$x' = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (3)$$

To show the results of normalization require showing the maximum value of data, and shows the minimum value of each attribute [21]. The min-max method provides data values with a range of values 0 and 1. Where 0 for data with minimum values and 1 for data that values maximum.

2) Matrix byte factor: The matrix byte factor process transforms categorical data into bytes, namely 0 and 1 [4]. Data factors are variables in the R-studio application to represent categorical data. Categorical data is used to categorize data as level or class data. In botnet encryption datasets, categorical data are data classes with two choices, namely botnet classes and standard classes. The data was transformed into byte form where 0 was for botnet class, and 1 was for standard class. Furthermore, the data is transformed into a matrix.

D. Data Model Distribution (Data Training and Data Testing)

TABLE III
DATA TRAINING AND DATA TESTING MODEL

No	Data Training (%)	Data Testing (%)
1	90	10
2	80	20
3	70	30
4	60	40
5	50	50

The process of distributing model data into training data and test data is carried out using a random sampling approach. So, each data has the chance to be selected as a subset of training data and test data. The data distribution process

model is made of five groups of data for training and testing, as shown in Table 3.

E. Model Architecture

The DNN model architecture is built using a hard library and the TensorFlow library in the R-Studio application. As explained in the model building environment, the hardness is one framework that is quite popular in the construction of DNN model architecture. The DNN model architects' construction process uses several parameters such as the input layer as input parameters, hidden layers, the number of neurons in each hidden layer, the activation function to activate input values in the hidden layer, and output values in the output layer. The DNN model architecture is built using the Rectified Linear Unit (ReLu) input activation function defined in equation (4) [3]. Meanwhile, the output layer using the sigmoid activation function is defined in equation (5) [18]. The DNN model architecture that is built is shown in Figure 3.

$$relu(x) = max[0, x] \tag{4}$$

$$y = f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

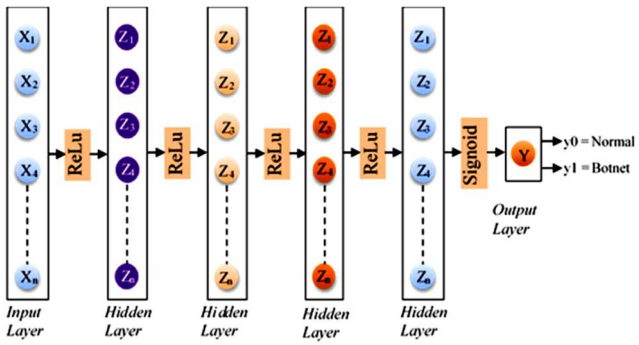

Fig. 3 Deep Neural Network (DNN) model architecture

F. Model Training

The DNN model architecture is built using a hard library and the TensorFlow library in the R-Studio application. As explained in the model building environment, the hardness is the training process uses the 0.01 learning rate model and five data group models in Table 3. Models are trained with nine scenarios. Each scenario uses a different number of params in each scenario, and the number of iterations in each param is also different. Params are the number of active neurons (input) multiplied by the number of weights plus the number of biases (biased in DNN even though they are not visible, but they remain) on each neuron in the hidden layer. The number of params used in the training model is 328, 653, and 1733 params shown in Table 4.

TABLE IV
PRAMS DEEP NEURAL NETWORK (DNN) MODEL

Layer	Input	Weight	Bias	Output	Number of Params
1-Hidden	11	11	11	(11,11,11)	(11*11) + 11 = 132
2-Hidden	15	15	15	(15,15,15)	(11*15) + 15 = 180
1-Output	15	1	1	(15,1,1)	(15*1) + 1 = 16
Total					132 + 180 + 16 = 328
3-Hidden	15	20	20	(15,20,20)	(15*20) + 20 = 320
Output	20	1	1	(20,1,1)	(20*1) + 1 = 21
Total					132 + 180 + 320+ 21 = 653
4-Hidden	20	50	50	(20,50,50)	(20*50) + 50 = 1050
1-Output	50	1	1	(50,1,1)	(50*1) + 1 = 51
Total					132 + 180 + 320 + 1050 + 51 = 1733

The Iterations used are 5, 10, and 20 epochs. The framework is popular in the construction of DNN model architecture. The DNN model architects' construction process uses several parameters such as the input layer as input parameters, hidden layers, the number of neurons in each hidden layer, the activation function to activate input values in the hidden layer, and output values in the output layer. The DNN model architecture is built using the Rectified Linear Unit (ReLu) input activation function defined in equation (4) [4]. Meanwhile, the output layer using the sigmoid activation function is defined in equation (5) [18]. The DNN model architecture that is built is shown in Figure 3. In some training scenarios, the model accuracy increases rapidly (see Table 5).

TABLE V
MODEL TRAINING ACCURACY AND LOSS

Param	Iteration	Data Training 90%		Data Training 80%		Data Training 70%		Data Training 60%		Data Training 50%	
		Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss
328	5 epoch	84.03	0.3927	88.93	0.3011	93.13	0.2418	85.21	0.355	68.6	0.6234
653	5 epoch	93.21	0.2354	93.14	0.2375	68.18	0.6259	68.37	0.6244	68.6	0.6229
1733	5 epoch	89.51	0.2403	93.06	0.1715	68.18	0.6262	68.37	0.6245	93.09	0.2524
328	10 epoch	90.04	0.2334	89.95	0.2408	92.98	0.2609	68.37	0.6238	68.6	0.6216
653	10 epoch	93.27	0.2307	90.01	0.232	93.11	0.2361	68.37	0.6242	93.22	0.157
1733	10 epoch	93.3	0.2318	88.98	0.2456	93.1	0.1564	93.06	0.1815	92.95	0.1789
328	20 epoch	90.02	0.2378	89.95	0.234	89.94	0.2649	88.38	0.2596	89.94	0.2342
653	20 epoch	93.78	0.2098	89.82	0.2341	93.04	0.1588	93.18	0.2354	89.95	0.2333
1733	20 epoch	88.97	0.2461	93.63	0.142	89.42	0.2408	68.37	0.4966	93.20	0.2147

Training data 90% accuracy increased from 84.03 - 93.78%. Training data model 80% accuracy 88.93 - 93.63%. Parameter iteration training data 90% increase significantly. Accuracy training data is 90% between 84.03-93.78%. Parameter iteration of the training data 90% increases significantly. The accuracy training data model is 80% between 88.93 - 93.63%.

The number of params and iterations in each training scenario also influences the model's accuracy from the two-

training data. Therefore, the accuracy of the second model of training data increases significantly in each training scenario. 90% and 80% of the training data did not show a significant decrease in inaccuracy. Conversely, it is different from the model on training data 70%, 60%, and 50%. The accuracy of the three-training data looks inconsistent. However, in specific scenarios with some params and different iterations, the models' average accuracy from the three-training data still looks relatively high. By the training model's objectives, the best accuracy models from the five training data groups in Table 4 was taken as the best model. The best model selection is taken based on the accuracy of the highest model and the smallest loss value, as shown in Table 6.

TABLE VI
BEST TRAINING MODEL

Data Training (%)	Param	Iteration	Accuracy (%)	*Loss*
90	653	20 *epoch*	93.78	0.2098
80	1733	20 *epoch*	93.63	0.142
70	1733	10 *epoch*	93.1	0.1564
60	1733	10 *epoch*	93.06	0.1815
50	653	10 *epoch*	93.22	0.157

The best training model in Table 6 is the model with the best accuracy and small loss. The five best models were taken from five different training data groups used in the model training process. Furthermore, the five models were tested and evaluated. The testing process uses test data divided in Table 3, and then the model was evaluated using matrix confusion in Figure 4.

		Prediction	
		Normal	*Botnet*
Actual	Normal	TN	FP
	Botnet	FN	TP

Fig. 4 Matrix confusion

G. Model Analysis and Evaluation

Using matrix confusion in Figure 4, the model was evaluated for the accuracy of detecting botnets using equation (6). Furthermore, the model was evaluated for the precision model level using equation (7) and the proportion of attacks that can be recovered or the ability of the model to rediscover botnet attacks on encryption using equation (8).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP+TN+FP+FN}} \times 100\% \quad (6)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \times 100\% \quad (7)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \times 100\% \quad (8)$$

True-Positive (TP) botnet class, correctly classified as a botnet class. False-negative (FN) botnet class, incorrectly classified as a normal class. True-negative (TN) normal class, correctly classified as a normal class. False-positive (FP) normal class, incorrectly classified as botnet class. Model evaluation is done by entering test data in matrix confusion showing the accuracy of detecting botnets on encryption traffic is relatively high. The model accuracy of each data testing in conducting botnet detection is relatively high (see Table 7).

TABLE VII
MODEL TEST AND EVALUATION RESULTS

Data Testing (%)	Total Data Testing	Class Target		TN	FP	FN	TP	Accuracy (%)	Recall (%)	Precision (%)
		Normal	Botnet							
10	2819	1036	1783	940	96	73	1710	94	95.91	94.68
20	5726	1996	3730	1730	266	33	3697	94.78	99.11	93.28
30	8569	2969	5600	2680	289	233	5367	93.91	95.83	94.89
40	11505	4043	7462	3655	388	294	7168	94.07	96.06	94.86
50	14472	5111	9361	5107	4	1146	8215	92.05	87.75	99.95

Table 7 shows that the accuracy and precision model levels detect botnets are high. However, the five models in separating botnet classes and standard information are still incompatible with the model produced by the model, seen in several models that detect botnet classes as standard classes and standard classes as botnet classes. Several test data model, namely 30%, 40%, and 50%, have a high degree of accuracy and precision, reaching 99.95%. However, errors in detecting botnet classes as standard classes are counted, namely 233, 294, and 1146 records. Because the error rate allows the model cannot be used as a good detection model. Contrast 10% test data model has the model 94-94.78% accuracy, and the precision is between 93.28-94.68%. In detecting botnet classes as standard classes, there are a few, namely 73 and 33 records. Compared with the three models, the model in the test data is 10%, and 20% is still quite good.

IV. CONCLUSION

The botnet detection model on encrypted traffic using the deep neural network (DNN) successfully detects botnets on well-encrypted traffic. The botnet detection model in the test data group of 20% is reasonably good in separating botnet classes with an accuracy of 94.78%.

REFERENCES

[1] R. Prasad and V. Rohokale, *Cyber Threats and Attack Overview*. 2020. doi: 10.1007/978-3-030-31703-4_2.

[2] F. Laurene, *Fundamentals of Neural Network, Architectures, Algorithm And Applications*. 1994. Accessed: Feb. 08, 2023. [Online]. Available: https://dl.matlabyar.com/siavash/Neural Network/Book/Fausett L.-Fundamentals of Neural Networks_ Architectures, Algorithms, and Applications (1994).pdf

[3] S. Chen, W. Sun, and W. Hu, "On dynamic hypervisor placement in virtualized software defined networks (vSDNs)," *International Conference on Transparent Optical Networks*, vol. 2020-July, pp. 1–5, 2020, doi: 10.1109/ICTON51198.2020.9203137.

[4] C.-L. L. Po-Wen Chi, Chien-Ting Kuo, He-Ming Ruan, Shih-Jen Chen, "An AMI threat detection mechanism based on SDN networks," *SECURWARE 2014 - 8th International Conference on Emerging Security Information, Systems and Technologies*, no. c, pp. 208–211, 2014, [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=securware_2014_9_30_30142.

[5] W. Li, W. Meng, and L. F. Kwok, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures," *Journal of Network and Computer Applications*, vol. 68, pp. 126–139, 2016, doi: 10.1016/j.jnca.2016.04.011.

[6] C. V. Neu, A. F. Zorzo, A. M. S. Orozco, and R. A. Michelin, "An approach for detecting encrypted insider attacks on OpenFlow SDN Networks," *2016 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016*, pp. 210–215, 2017, doi: 10.1109/ICITST.2016.7856698.

[7] H. L. Chen Jing, Cheng Xi, Du Ruiying and W. Chiheng, "BotGuard: Lightweight real-time botnet detection in software defined networks," *Wuhan University Journal of Natural Sciences*, vol. 22, no. 2, pp. 103–113, 2017, doi: 10.1007/s11859-017-1223-8.

[8] R. Hadianto and T. W. Purboyo, "A Survey Paper on Botnet Attacks and Defenses in Software Defined Networking," *International Journal of Applied Engineering Research*, vol. 13, no. 1, pp. 483–489, 2018, [Online]. Available: http://www.ripublication.com

[9] S. C. Su, Y. R. Chen, S. C. Tsai, and Y. B. Lin, "Detecting P2P Botnet in Software Defined Networks," *Security and Communication Networks*, vol. 2018, 2018, doi: https://doi.org/10.1155/2018/4723862.

[10] D. S. Rana, S. A. Dhondiyal, and S. K. Chamoli, "Software Defined Networking (SDN) Challenges, issues and Solution," *International Journal of Computer Sciences and Engineering*, vol. 7, no. 1, pp. 884–889, 2019, doi: 10.26438/ijcse/v7i1.884889.

[11] Manoj Kumar Putchala B.E., "Deep Learning Approach for Intrusion Detection System (Ids) in the Internet of Things (Iot) Network Using Gated Recurrent Neural Networks (Gru)," *Thesis*, vol. 1, no. 1, pp. 1188–1197, 2017, [Online]. Available: https://corescholar.libraries.wright.edu/etd_all/1848/.

[12] P. K. Hamza Mutaher, Abdul Wahid, "Openflow Controller-Based SDN:Security Issues and Countermeasures," *International Journal of Advanced Research in Computer Science*, vol. 9, no. 2, pp. 397–401, 2018, doi: http://dx.doi.org/10.26483/ijarcs.v9i1.5498.

[13] S. Singaravel, J. Suykens, and P. Geyer, "Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction," *Advanced Engineering Informatics*, vol. 38, no. May, pp. 81–90, 2018, doi: 10.1016/j.aei.2018.06.004.

[14] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in SDN home gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018, doi: 10.1109/ACCESS.2018.2872430.

[15] F. Chollet, *Deep Learning with Python*. 2018, Manning Publications, 2018. [Online]. Available: http://faculty.neu.edu.cn/yury/AAI/Textbook/Deep Learning with Python.pdf.

[16] U. Wijesinghe, U. Tupakula, and V. Varadharajan, "Botnet detection using software defined networking," *22nd International Conference on Telecommunications (ICT 2015)*, no. Ict, pp. 219–224, 2015, doi: 10.1109/ict.2015.7124686.

[17] S. Gnanambal, "Classification Algorithms with Attribute Selection : an evaluation study using WEKA," *Int. J. Advanced Networking and Applications*, vol. 3644, pp. 3640–3644, 2018, [Online]. Available: http://oaji.net/articles/2017/2698-1528114152.pdf.

[18] K. Kim and M. E. Aminanto, "Deep learning in intrusion detection perspective: Overview and further challenges," *Proceedings - WBIS 2017: 2017 International Workshop on Big Data and Information Security*, vol. 2018-Janua, pp. 5–10, 2018, doi: 10.1109/IWBIS.2017.8275095.

[19] G. Vormayr, T. Zseby, and J. Fabini, "Botnet Communication Patterns," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2768–2796, 2017, doi: 10.1109/COMST.2017.2749442.

[20] S. Gaonkar, N. F. Dessai, J. Costa, A. Borkar, S. Aswale, and P. Shetgaonkar, "A Survey on Botnet Detection Techniques," *International Conference on Emerging Trends in Information Technology and Engineering, ic-ETITE 2020*, pp. 1–6, 2020, doi: 10.1109/ic-ETITE47903.2020.Id-70.

[21] C. Yin, Y. Zhu, J. Fei, and X. He, "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017, doi: 10.1109/ACCESS.2017.2762418.