An Artificial Deep Neural Network for the Binary Classification of Network Traffic

Shubair A. Abdullah[1]
College of Education, Department of Instructional and learning Technology, Sultan Qaboos University
Muscat, Oman

Ahmed Al-Ashoor[2]
College of Pharmacy, Department of Pharmacognosy
University of Basra
Basra, Iraq

***Abstract*—Classifying network packets is crucial in intrusion detection. As intrusion detection systems are the primary defense of the infrastructure of networks, they need to adapt to the exponential increase in threats. Despite the fact that many machine learning techniques have been devised by researchers, this research area is still far from finding perfect systems with high malicious packet detection accuracy. Deep learning is a subset of machine learning and aims to mimic the workings of the human brain in processing data for use in decision-making. It has already shown excellent capabilities in dealing with many real-world problems such as facial recognition and intelligent transportation systems. This paper develops an artificial deep neural network to detect malicious packets in network traffic. The artificial deep neural network is built carefully and gradually to confirm the optimum number of input and output neurons and the learning mechanism inside hidden layers. The performance is analyzed by carrying out several experiments on real-world open source traffic datasets using well-known classification metrics. The experiments have shown promising results for real-world application in the binary classification of network traffic.**

***Keywords*—Deep learning; ANN; packet classification; binary classification; malicious traffic classification**

I. Introduction

The classification of network packets refers to the task of identifying abnormal behavior in networks. Currently, governmental and organizational networks across the world are natural targets for attackers who aim to compromise them in order to perform illegal activities such as information stealing. As each generation of malware is progressively more advanced, the development of successful online intrusion detection systems is at the forefront of information security tasks.

Naturally, a host compromised by malware will most probably generate packets that serve the malware's activities, i.e. malicious packets. A packet is a container used to carry data over a network. It normally represents the smallest amount of data that can traverse over a network at a single time. Normal TCP/IP packets contain several forms of information, including the data it is carrying, source and destination IP addresses, source and destination port numbers, and other information related to the quality of service and packet handling. A straightforward way to detect intrusions is with packet classification, which could be implemented using machine learning techniques. Machine learning is an application of artificial intelligence that provides software with the ability to automatically learn and evolve from experience without being explicitly programmed. It could be used to solve problems of predictions and classifications. In general, machine learning techniques are divided into two types: supervised and unsupervised. Packet classification is modeled as classification problem in supervised learning. Supervised learning has a set of input features and output classes. It has an algorithm to learn the mapping function from the input features to the output class. The goal is to approximate the mapping function. When new input features are introduced, the algorithm predicts the output class. In unsupervised learning, on the other hand, there is a set of input features without corresponding output classes to perform the learning task. The goal for unsupervised learning is to model the underlying distribution in input data to learn more about the data [1].

Deep learning is a subset of machine learning. In a deep learning system, multiple layers, i.e. input, hidden, and output layers, are stacked to form a neural network. Each hidden layer applies neuron mathematical structures to perform the learning task. The learning approach is designed to analyze data continually with a logic structure similar to how a human would draw a conclusion. The data analysis is repeated as long as inaccurate predictions occur. When the system returns a low accurate prediction level, the learning approach will automatically make an adjustment. Usually, a deep learning neural network has more than one hidden layer, which determines the network depth between the input and output layers. The learning process consists of two crucial elements: forward feature abstraction and backward error feedback. The first element is important for input data analysis and the second is important for tweaking the neurons [2].

Two gaps were observed in the literature related to the classification of the network packet problem, though there is a long record of research in packet classification over the last three decades [3]. The first gap is that the research field is still far from finding a perfect system with high malicious packet detection accuracy. The second gap is the lack of comprehensive research attempts that have employed deep learning approaches to classify network packets. Since it is a relatively new research area, there have been few research attempts investigating, evaluating, and tuning well-known deep learning approaches to classify the network packet problem [4].

The main contribution of this paper is to fill the abovementioned gaps by designing and implementing an artificial deep neural network (ADNN) using the state-of-theart methodologies of deep neural networks. The ADNN is evaluated using standard classification quality metrics and compared with well-known classification algorithms including kNN, SVM, and Naive Bayes.

The rest of the article is divided into five sections. Section II reviews notable research in the area. Section III provides the architectural designs of the ADNN proposed in this study along with the implementation details, including the hardware and software used. Section IV presents the experimental results. Finally, Section V concludes the paper and presents some future facts.

II. Related Work

Several research papers have been published in the last decades dealing with enhancing the performance of network packet classifications. Most published papers have employed both supervised and unsupervised machine learning approaches. Examples of supervised approaches employed include support vector machine (SVM) [5] and k-nearest neighbors (kNN) [6]. For unsupervised approaches, the most common employed approach is k-means clustering [7]. Interested readers may refer to the work of Nguyen et al. [8] and Dainotti et al. [9] for a detailed overview of the machine learning techniques applied to traffic classification. Abdullah et al. [10] proposed a novel evolving fuzzy system to discriminate anomalies by inspecting the network traffic. The system incorporated the knowledge base-evolving mechanism and showed a significant positive impact on the classification accuracy. An open source tool for network traffic classification called the traffic identification engine (TIE) was developed in 2008 and gradually evolved over the years from 2009 to 2014 through the support of the open source community. TIE uses a combination of different traffic classification techniques and can be applied to both live traffic and previously captured traffic traces [11].

The application of deep neural network approaches such as deep autoencoders, deep belief neural (DBN) networks, deep convolutional neural networks (CNN), and recurrent neural networks (RNN) to solve the packet classification problem is a relatively new area of research. These approaches have already shown excellent capabilities in dealing with real-world problems such as facial recognition [12], intelligent transportation systems [13], etc. Lotfollahi et al. [14] proposed the “deep packet” system employing the deep CNN approach to integrate feature extraction and classification. Deep packet can handle traffic characterization to categorize network traffic into classes, i.e. FTP and P2P, and application identification to identify end-user application e.g. BitTorrent and Skype. Rahul [15] applied deep learning techniques to the classification of network protocols and applications using flow features and data signatures. They used their own dataset for traffic identification and the Microsoft Kaggle dataset for malware classification tasks. The DBN network is a type of generative neural network that uses an unsupervised machine learning model to produce results. Alom [16] explored the capabilities of the DBN network in performing intrusion detection. They performed a series of experiments after training the DBN network with the NSL-KDD dataset. The RNNs are designed for sequence prediction problems, which involve using historical sequence information to predict the next values or next single value in a sequence. Lopez-Martin et al. [17] presented a complete study on several architectures that integrate a CNN and an RNN. They showed that the integration of RNN with CNN could provide the best results for the Internet of Things (IoT) network traffic classification.

Despite these efforts, the literature lacks comprehensive attempts that have investigated, evaluated, and tuned well-known deep learning approaches for classifying network packets. The strategy used in the research methodology in this paper is to investigate and experiment each stage in building the network separately. Moreover, the final stage involves tuning the parameters in order to reach the highest possible level of accuracy in classifying network packets into malicious and normal packets.

III. Methodology

A. Dataset Description

Suppose that D is a supervised training dataset for network packet classification with i-tuple elements. D is divided into two subsets: DN contains normal packets and DM contains malicious packets:

$$D_N \subset D \wedge D_M \subset D \leftrightarrow D \equiv D_N \cup D_M$$

The D set can be represented by the set builder notation:

$$D = \{x | x \in D_N \vee x \in D_M\}$$

Where x is 11-tuple element that includes 10 features plus 1 class for describing the packet. The class of the packet c is defined as follows:

$$c = \begin{cases} 0 \ if \ x \in D_N \\ 1 \ if \ x \in D_M \end{cases}$$

In a normal situation, i.e. where there are no malicious packets, packets are considered as normal, that is $X_N = \{x_1, x_2, x_3, x_n\} \subset D_N$. This situation can be represented as follows:

Let P(x) denote x $\in X_N$ where $X_N \subset D_N$

Then the truth-value of $\forall\, x\, P(x)$ is True (1)

The truth-value of (1) is changed to false if the universe of discourse contains normal packets and malicious packets, that is $X_{NM} = \{x_1, x_2, x_3, x_n\}$, $X_{NM} \subset D_N \wedge X_{NM} \subset D_M$. This new situation is represented as follows:

Let P(x) denote x $\in X_{NM}$ where $X_{NM} \subset D_N \cup D_M$

Then the truth-value of $\forall\, x\, P(x)$ is False (2)

The truth-value of $\exists x\, P(x)$ is True (3)

In this case x is called a counterexample for (2) since it turns its truth-value into false. The objective of the ADNN is to identify and classify the counterexamples as malicious.

The dataset used in this research was prepared from the UNSW-NB 15 dataset, which has been created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) in the University of New South Wales, Canberra, Australia [18]. It contains a hybrid of real modern normal activities and synthetic contemporary attack behaviors. The training set contains 175,341 records and the testing set contains 82,332 records from both normal and malicious packets. For each record, there are 48 features (dependent variables) and one label (dependent variable). The preparation of training and test datasets involved.

1) Feature selection: this task aims to find and select the most useful features in a dataset. The features with low importance are removed. For example, the feature *swin*, which refers to the value of source TCP window advertisements and the feature *dwin*, which refers to the value of destination TCP window advertisements.

2) Encoding categorical features: this task aims to convert categorical features into numeric values. Three categorical features exist in the dataset: proto, service, and state. The values in these features have no ordinal relationship. Therefore, integer encoding was used [2]. Each unique category value was assigned an integer value. Table I describes the features that are selected.

3) Dataset filtration: this task is done by removing the records with a high percentage of missing values. For example, any record with a service marked as (- hyphen) or with duration equal to zero was removed from the dataset. Table II describes the dataset after completing the filtration task.

4) Feature scaling: this task is important when working with a learning model. It aims to scale the features to a range centered on zero. It prevents features that have high variance from dominating other features in the dataset. The standard scaler and the results were as expected and the features are normalized so that they have mean = zero and standard deviation = one.

B. *ADNN Architecture*

The architecture of the ADNN is shown in Fig 1. It is composed of four layers, namely an input layer, two hidden layers, and an output layer.

TABLE. I. DESCRIPTION OF 12-TUPLE ELEMENTS (PACKET FEATURES)

#	Feature	Description
1	dur	Record total duration
2	proto	Transaction protocol (TCP | UDP)
3	service	0=http, 1=ftp, 2=smtp, 3=ssh, 4=dns, 5=ftp-data , 6=irc and 7=(-) if not a much used service
4	state	Indicates the state and its dependent protocol, e.g. 0=ACC, 1=CLO, 2=CON, 3=ECO, 4=FIN, 5=REQ, and 6=RST
5	spkts	Source to destination packet count
6	sbytes	Source to destination transaction bytes
7	sttl	Source to destination time to live value
8	sload	Source bits per second
9	swin	Source TCP window advertisement value
10	synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets
11	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets
12	label	0= normal and 1=malicious

TABLE. II. DATASET DESCRIPTION

Item	Description
Total number of records	210,191 records
Normal packets	124,709 records
Malicious packets	85,482 records

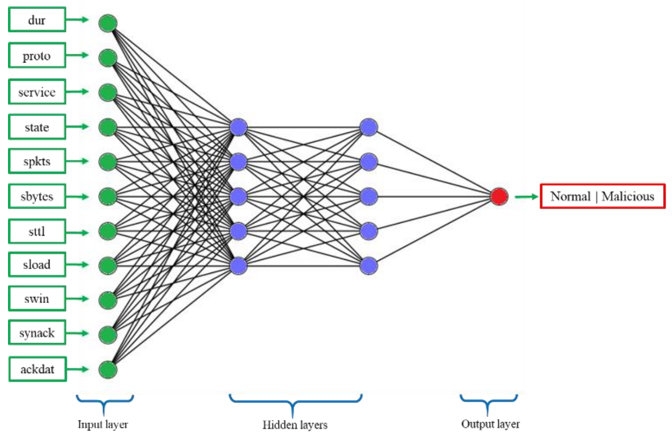


Fig. 1. ADNN Architecture.

The input layer is the first layer of the ADNN. It does not apply any operations and has no associated values of weights. It consists of 11 neurons, one neuron for each input feature. Given a set of training samples $\{x_1, x_2, x_3, \dots x_n\}$, where $x_i \in X_N \vee x_i \in X_M$, an input neuron accepts x_i and passes it to one or more neurons in the next layer - the first hidden layer.

Two hidden layers were created in the ADNN and each layer contains five neurons. All the neurons are connected to every neuron in the next layer. For each neuron, there are a certain number of inputs and weights. The number of weights for a neuron equals the number of its input values. Each neuron in hidden layer #1 has 11 inputs and 11 weights, and each neuron in hidden layer #2 has five inputs and five weights. Weights are crucial to ADNN functioning because they are learnable parameters. The values of weights are initialized randomly to be close to zero but not zero before the learning starts. When presented with data during training, their values are adjusted to new values, and this adjustment will contribute to deciding the importance of inputs.

Three operations are done by a single neuron. First, it calculates the weighted summation of all the input values (x_n). Then, it applies an activation function to the weighted summation. Finally, it passes the results to a neuron in the next layer, as shown in:

$$\acute{Z} = \sum_{i=1}^{m}(w_i x_i) \tag{4}$$

$$\bar{O} = \emptyset(\acute{Z}) \tag{5}$$

Where w_i is an input data x_i weight, m is the number of neuron input data, $\acute{Z}$ is the weighted summation, $\bar{O}$ is the output of the neuron, and Ø *(theta)* is the activation function.

The activation function is responsible for transforming the weighted summation from the neuron into the activation of the next neuron. There are several activation functions in the literature. In this study, we used the rectified linear activation unit, or ReLU for short, for two reasons: (1) its computational simplicity and (2) its linear behavior increases the chances of optimizing the ADNN [19]. The ReLU activation function is formalized as below:

$$\emptyset(\acute{Z}) = max(\acute{Z}, 0) \qquad (6)$$

This is the last layer in the ADNN. It receives input from hidden layer #2, makes some transformation, and outputs a binary (zero = normal or one = malicious). It consists of a single neuron that calculates the weighted summation of its input values and applies the sigmoid activation function to produce the final output. As we have two events that are mutually exclusive and cannot both occur at the same time (normal traffic and malicious traffic), we used the sigmoid activation function, which performs perfectly in this type of classification problem. Moreover, a single sigmoid neuron can be used to estimate the probability p(y=1) [19]. The sigmoid activation function is represented as below:

$$\bar{y} = \emptyset_1(\acute{Z}) = \frac{1}{1+e^{-\acute{Z}}} \qquad (7)$$

Where $\acute{Z}$ is the output of hidden layer #2 calculated as in (5) and (6), $\bar{y}$ is the output of the neuron, and $\emptyset_1$ is the sigmoid activation function. Fig. 2 illustrates the architecture of the hidden and output layers.

The input $\{x_1, x_2, x_3, \dots x_n\}$, where $x_i \in X_N \vee x_i \in X_M$ provides the initial information that propagates to the hidden neurons at each layer and finally produces the output $\bar{y}$, which is a number in the range from 0 – 1. We used the cross-entropy loss function [2] to compute the average error across all examples. The cross-entropy loss function is represented as follows:

$$H(y, \bar{y}) = -\sum_{i=1}^{n} y_i \log \bar{y}_i$$

Where y is the actual value, $\bar{y}$ is the output of the ADNN, and $H(y, \bar{y})$ is the cross-entropy loss function. After each forward propagation, the ADNN seeks a set of weights that minimize the difference between $\bar{y}$ and y. To get the least possible difference, the ADNN backpropagates the information about the error through the layers in order to tweak the weights and recalculate a new $\bar{y}$. We used the adaptive moment estimation (Adam) optimizer [20], which is a search technique to tweak weights in each neuron in the hidden layers. Adam is an adaptive learning rate optimizer that has been designed specifically for training deep neural networks. There are other options for optimizing the weights of neurons i.e. root mean square propagation (rmsprop), which is a gradient-based optimization technique.

In deep learning, when an entire dataset is passed forward and backward through the neural network once, this full cycle is called an epoch. The number of epochs is a tunable parameter, and usually more than one epoch is used. To optimize the learning, we used 20 epochs to train the ADNN. The batch size, which is the number of training examples in one epoch, is set to 10 samples in order to avoid overloading the processor and the RAM of the computer.

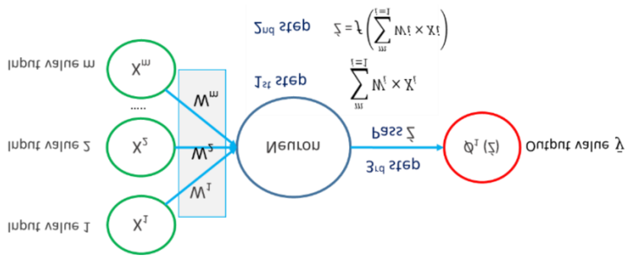


Fig. 2. Neuron of Hidden and Output Layers.

IV. Experiments

The experiments conducted aligned with the strategy of the research methodology. Five experiments were conducted aiming to gradually and systematically build and optimally set up the ADNN. At each experiments, an investigation task for a deep learning technique was performed.

Prominent metrics were used to evaluate the classification quality of the ADNN, such as accuracy, area under curve (AUC), recall, precision, and F1. These evaluation metrics were computed using a confusion matrix, which presents four measures: True Positive (TP): malicious traffic is classified by the ADNN as malicious traffic; False Positive (FP): normal traffic is classified by the ADNN as malicious traffic; True Negative (TN): malicious traffic is classified by the ADNN as normal traffic; False Negative (FN): normal traffic is classified by the ADNN as normal traffic.

A. *Initial Experiment*

The initial experiment was conducted in a straightforward way only to verify the code implementation and the parameter configurations. The dataset was split randomly into 75% training set (157,643 samples) and 25% test set (52,548 samples). The values for the number of epochs, batch size, and optimizer are 20, 10, and 'Adam' respectively. Fig. 3 shows the results of fitting the ADNN to the training set. It shows the accuracy for each epoch. The accuracy of the first epoch is 81%, which then increases steadily until it reaches a peak of 86% in the eleventh epoch. The accuracy begins to stabilize at slightly below 85% in the fourteenth epoch. The mean value of 84% along with the variance 0.01 show that the accuracies of 20 epochs are related to each other.

Figure 4 shows the results of classifying the test set in terms of the confusion matrix. The accuracy is calculated using equation, and te result generated is 84%:

$$accuracy = (tp + tn)/(tp + tn + fn + fp)$$

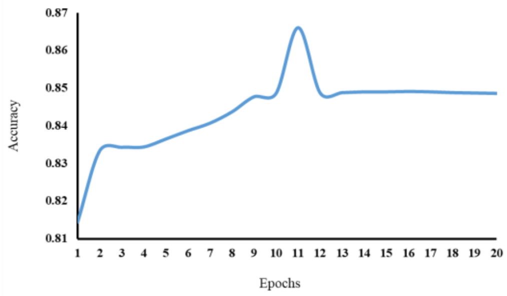
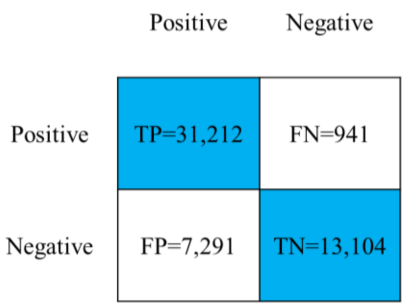


Fig. 3. Epoch Accuracies for Fitting the Training Set.



	Positive	Negative
Positive	TP=31,212	FN=941
Negative	FP=7,291	TN=13,104

Fig. 4. Confusion Matrix for Classifying the Test Set in the First Experiment.

B. K-Fold Cross-Validation Technique

In the previous experiment, the ADNN was trained using a 75% training set (157,643 samples), with the result revealing a low accuracy both for the training set and test set. Judging the ADNN performance on the accuracy obtained from one test set does not give a complete idea of the performance with regard to variance. Variance occurs when very different accuracies are obtained after testing a model using different test sets. In order to optimize the method used to evaluate the ADNN, the k-fold cross-validation technique is employed in this experiment. The advantage of this technique is that all samples are used for both training and validation, with every single sample being used for validation exactly once. The following steps were followed in this experiment:

1) The original dataset comprised of 210,191 samples was randomly partitioned into 10 equal sized subsets. Each subset contained 12,019 samples. k=10 was chosen as it is commonly used in the literature.

2) The partitioning of the original dataset into 10 subsets was governed by criteria to ensure that each subset has 60% normal samples and 40% malicious samples. A stratified cross-validation process that is common variation of cross-validation to ensure each subset has the same proportion of normal and malicious samples was used. We used a 60/40 proportion to create a semi-stratified cross-validation.

3) Of the 10 subsets, a single subset was used to testing. The remaining 9 subsets were used as the training sets.

4) The cross-validation process was repeated 10 times, with each subset being used only one time as the test set.

5) The values of the epochs and batch size variables used in the previous experiment were used again.

To calculate the accuracy, the 10 results were averaged. Fig. 5 shows the results of the semi-stratified 10-fold cross-validation experiment. The total number of samples was 210,191. In each of the 10 folds, there were 189,172 samples as the training set and 21,019 samples as the test set. In contrast with the accuracy of 84% obtained in the training phase of the previous experiment, the mean accuracy of 86% reflects an improvement in the building of the ADNN. The resulting low variance of 0.004 also suggests an improvement in the ADNN.

To determine the accuracy of the ADNN precisely, a test set of 50,000 unseen samples was prepared to test the ADNN performance on unseen samples. Fig. 6 shows the results of classifying the test set in terms of the confusion matrix. An accuracy of 84% was calculated - the same value calculated in the previous experiment. Although the variance obtained in the training phase is quite low, the accuracy obtained from the test phase indicates the presence of a bias. The low accuracy means that there is a difference between the average prediction of the ADNN and the correct value.

C. Dropout Technique

The challenge was to beat the low accuracy of 84% obtained from testing the ADNN on the unseen 50,000 samples in the previous experiment. The accuracy obtained from the training part of 86% was probably the result of using a small dataset, which may cause overfitting and poor performance. When the ADNN was faced with the unseen 50,000 samples, it predicted them with lower accuracy than in the training. In such a situation, there is a need for regularization. Dropout is an approach to regularize deep neural networks that helps reducing interdependent learning amongst the neurons [21]. It refers to dropping out randomly selected neurons from a certain layer during the training. Consequently, the outputs of the dropped neurons are not considered during a particular forward or backward pass. Normally, the dropout technique is applied on the hidden layers and has been proven to enhance the performance of deep neural networks over other regularization methods [22]. In this experiment, one neuron from the hidden layers was dropped. The stratified 10-fold cross-validation was implemented on the same dataset used in the previous experiment, with 189,172 samples as the training set and 21,019 samples as the test set in each fold. Fig. 7 shows the results after applying the dropout technique on the hidden layers. The mean accuracy of 90% obtained reflects an encouraging improvement in building the ADNN.

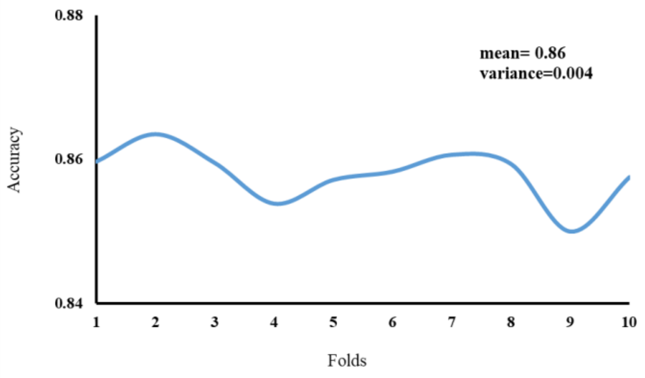
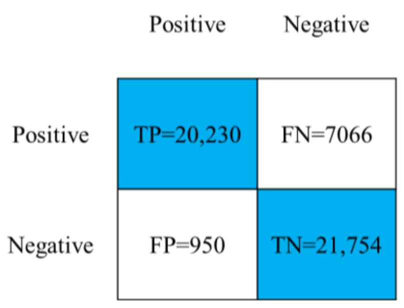


Fig. 5. 10-Fold Cross-Validation Accuracies for Fitting the Training Set.



	Positive	Negative
Positive	TP=20,230	FN=7066
Negative	FP=950	TN=21,754

Fig. 6. Confusion Matrix for Classifying the Test Set in the Second Experiment.

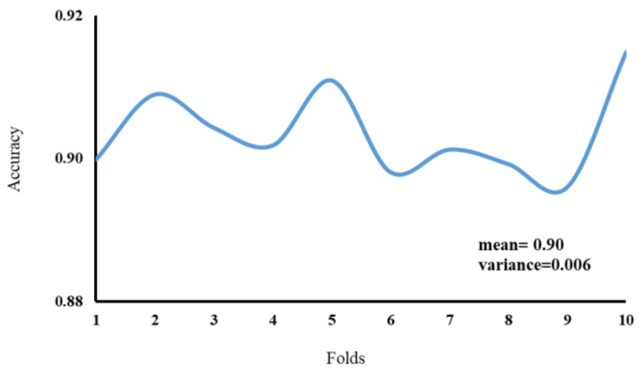


Fig. 7. 10-Fold Cross-Validation Accuracies for Fitting the Training Set after Applying the Dropout Technique.

To verify the ADNN performance on unseen samples, a test was conducted using the same test set used in the previous experiment (50,000 samples). Figure 8 shows the confusion matrix resulting from the test. The accuracy of 90% obtained confirms the enhancement in the ADNN after applying the dropout technique.

D. Parameter Tuning

Despite the improvements achieved in building up the ADNN, there was still room to enhance the prediction accuracy. The best tool to use to achieve a higher accuracy than 90% at this stage was parameter tuning. The ADNN has two types of parameters: 1) tweaking parameters, i.e. the weights learned from the model during the training and 2) hyperparameters, i.e. number of epochs, batch size, the optimizer, and the number of neurons in the layers.

The technique of grid-search cross-validation (GSCV) [2] was used to find the optimal hyperparameters of a neural network that result in the most accurate prediction. The GSCV technique tests several combinations of hyperparameters values and returns the best selection choice that leads to the best accuracy. The GSCV technique usually takes a long time to test the values and can be computationally expensive in case of huge dataset and the number of hyperparameters to be tuned is large. To avoid this, the training phase involved only three hyperparameters, the number of epochs, the batch size, and the optimizer. Table III describes the hyperparameters and the combinations of values that are tested. The number of neurons, number of folds (k), and hidden layers were not changed. The accuracy obtained is 91% for fitting the ADNN to the training set.

	Positive	Negative
Positive	TP=21,547	FN=4,429
Negative	FP=801	TN=23,223

Fig. 8. Confusion Matrix for Classifying the Test Set in the Third Experiment.

TABLE. III. HYPERPARAMETER TUNING

Hyperparameter	Values tested	Best value
Number of epochs	30 and 35	35
Batch size	25 and 32	32
Optimizer	"Adam" and "rmsprop"	"rmsprop"

	Positive	Negative
Positive	TP=21,855	FN=3,694
Negative	FP=761	TN=23,690

Fig. 9. Confusion Matrix for Classifying the Test Set in the Fourth Experiment.

In the test phase, the same test set (50,000 samples) was used as in the previous experiments, with the accuracy resulting from parameter tuning found to be 91%. Fig 9 shows the confusion matrix resulting from the test.

E. Imbalance Classification Problem

The imbalance classification problem occurs in binary classification when the rate of one class is outnumbered by the other class. Two classes were used in this malicious packet classification, namely normal packets and malicious packets, with the former representing the majority of the dataset. In such a situation, the accuracy is not an optimum measure for assessing the ADNN performance. Two characteristics of the ADNN performance were assessed. First, the ADNN's ability to classify the malicious packets, which are the packets of interest in the dataset. Second, the proportion of packets that the ADNN classifies as malicious that indeed are actually malicious. Recall and precision metrics were used to assess the two characteristics. The recall and precision metrics were identified as follows:

$$recall = \frac{tp}{tp+fn} \quad , \; precision = \frac{tp}{tp+fp}$$

There is a tradeoff between the precision and the recall in binary classification. As the precision increases, the recall decreases and vice-versa. Finding an optimal balance of recall and precision was achieved by combining the two metrics using the F_1 score, which is a harmonic mean of precision and recall that summarizes the model's ability for a specific probability threshold (0.5). The F_1 score was computed as in the following equation.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

The precision-recall curve (PRC) metric was used to summarize the model's performance across more than one threshold. The PRC is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds. Instead of illustrating the curves, the area under the curve (AUC) is calculated. The AUC is an integral summary of the model's performance. A model that performs perfectly has an AUC of 1.0. We compared the accuracy, AUC, recall, precision, and F_1 scores with scores of three commonly used models in the literature: kNN, SVM, and Naïve Bayes. The scores for testing the 52,548 samples with both ADNN and conventional machine learning models are shown Table IV. Fig. 10 shows the TP, FP, FN, and TN scores for ADNN and the three machine learning models. The results show that ADNN is the superior method in terms of accuracy.

TABLE. IV. ACCURACY, AUC, RECALL, PRECISION, AND F1 RESULTS

Algorithm	**Acc**	**AUC**	**Recall**	**Pre**	**F_1**
ADNN	0.92	0.86	0.61	0.96	0.75
kNN (k=5)	0.87	0.83	0.61	0.90	0.73
SVM (kernal='rbf')	0.91	0.85	0.60	0.91	0.72
Naive Bayes	0.89	0.85	0.61	0.86	0.71

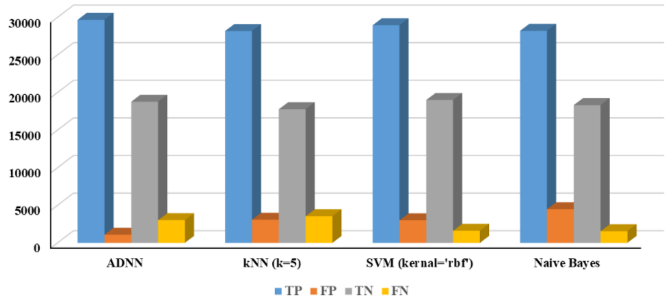


Fig. 10. TP, FP, FN, and TN Results.

V. CONCLUSION

An artificial deep neural network for binary classifying network packets into malicious and normal packets was presented in this paper. The strategy for building up the deep neural network followed systematic stages in order to reach the highest possible level of accuracy. In each stage, an investigation task for a deep learning technique was performed, followed by experiments involving the technique itself. In the final stage, the parameters of the neural network were tuned to confirm the optimum setup. For training and evaluation of the artificial deep neural network, the UNSW-NB dataset was used. The UNSW-NB dataset was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security. The preparation of training and test datasets involved four tasks: feature selection, encoding of categorical features, dataset filtration, and feature scaling. The performance was compared with three commonly used models in the literature: kNN, SVM, and Naïve Bayes. The results show that the artificial deep neural network is superior to the competing models in terms of accuracy. Our future research will be directed towards investigating other classes of deep neural networks, e.g. DBN, RNN, and CNN, and applying these algorithms on different public traffic data sets to examine their effectiveness.

REFERENCES

[1] Luo, M., Wang, L., Zhang, H., Chen, J.: A research on intrusion detection based on unsupervised clustering and support vector machine. In: International Conference on Information and Communications Security 2003, pp. 325-336. Springer.

[2] Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press, Cambridge, MA, USA (2016).

[3] lizadeh, H., Zúquete, A.: Traffic classification for managing Applications' networking profiles. Security and Communication Networks 9(14), 2557-2575 (2016).

[4] Naseer, S., Saleem, Y., Khalid, S., Bashir, M.K., Han, J., Iqbal, M.M., Han, K.: Enhanced network anomaly detection based on deep neural networks. IEEE Access 6, 48231-48246 (2018).

[5] Este, A., Gringoli, F., Salgarelli, L.: On-line SVM traffic classification. In: 2011 7th International Wireless Communications and Mobile Computing Conference 2011, pp. 1778-1783. IEEE.

[6] Li, W., Yi, P., Wu, Y., Pan, L., Li, J.: A new intrusion detection system based on KNN classification algorithm in wireless sensor network. Journal of Electrical and Computer Engineering 2014 (2014).

[7] Zhang, J., Xiang, Y., Zhou, W., Wang, Y.: Unsupervised traffic classification using flow statistical properties and IP packet payload. Journal of Computer and System Sciences 79(5), 573-585 (2013).

[8] Nguyen, T.T.T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys & Tutorials 10(4), 56-76 (2008). doi:10.1109/SURV.2008.080406.

[9] Dainotti, A., Pescape, A., Claffy, K.C.: Issues and future directions in traffic classification. IEEE network 26(1), 35-40 (2012).

[10] Shubair, A., Al-Hashmi, A.S.: TiSEFE: Time Series Evolving Fuzzy Engine for Network Traffic Classification. International Journal of Communication Networks and Information Security 10(1), 116-124 (2018).

[11] Donato, W.D., Pescape, A., Dainotti, A.: Traffic identification engine: an open platform for traffic classification. IEEE Network 28(2), 56-64 (2014). doi:10.1109/MNET.2014.6786614.

[12] Sun, Y., Chen, Y., Wang, X., Tang, X.: Deep learning face representation by joint identification-verification. In: Advances in neural information processing systems 2014, pp. 1988-1996.

[13] Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.-Y.: Traffic flow prediction with big data: a deep learning approach. IEEE Transactions on Intelligent Transportation Systems 16(2), 865-873 (2015).

[14] Lotfollahi, M., Zade, R.S.H., Siavoshani, M.J., Saberian, M.: Deep packet: A novel approach for encrypted traffic classification using deep learning. arXiv preprint arXiv:1709.02656 (2017).

[15] Rahul, R., Anjali, T., Menon, V.K., Soman, K.: Deep learning for network flow analysis and malware classification. In: International Symposium on Security in Computing and Communication 2017, pp. 226-235. Springer.

[16] Alom, M.Z., Bontupalli, V., Taha, T.M.: Intrusion detection using deep belief networks. In: 2015 National Aerospace and Electronics Conference (NAECON) 2015, pp. 339-344. IEEE.

[17] Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., Lloret, J.: Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. IEEE Access 5, 18042-18050 (2017). doi:10.1109/ACCESS.2017.2747560.

[18] 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) 2015, pp. 1-6. IEEE.

[19] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics 2011, pp. 315-323.

[20] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).

[21] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929-1958 (2014).

[22] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012).