Intrusions Detection and Classification Using Deep Learning Approach

Léonard M. Sawadogo(✉), Didier Bassolé, Gouayon Koala, and Oumarou Sié

Laboratoire de Mathématiques et d'Informatique, Université Joseph KI-ZERBO, Ouagadougou, Burkina Faso
http://www.univ-ouaga.bf

Abstract. In this paper we propose an intrusions detection technique using Deep Learning approach that can classify different types of attacks based on user behavior and not on attacks signatures. The Deep Learning approach used is Supervised Learning model called Convolutional Neural Networks (CNN) coupled with Tree Structure whose set is named Tree-CNN. This structure allows for incremental learning. This makes the model capable of learning how to detect and classify new types of attacks as new data arrives. The model was implemented with TensorFlow and trained with the CSE-CIC-IDS2018 dataset. We evaluated the performance of our proposed model and we made comparisons with other approaches considered in related works. The experimental results show that the model can detect and classify intrusions with a score of 99.94% for the detection and 97.54% for the classification.

Keywords: Intrusion detection · Deep learning · Classification · Tree-CNN

1 Introduction

Intrusions generally involve gaining unauthorized access to data on a computer system or network by bypassing or defusing the security devices in place. Intrusion detection systems for the security of computer systems are diversified, but they are still confronted with two major problems, namely the rate of false alarms and the capacity to detect new attacks, in particular "zero-day" attacks. The goal of intrusion detection is to spot the actions of an attacker attempting to take advantage of system vulnerabilities to undermine security objectives. Different categories of intrusion detection methods are explored in the literature, in particular those based on a behavioral approach such as static analysis, Bayesian analysis, neural networks and those based on a scenario approach such as the search for signatures, pattern matching, simulation of Petri networks.

An intrusion detection system can attempt to identify attacks by relying on information relating to transitions taking place in the system (execution of certain programs, certain sequences of instructions, arrival of certain packets

Y. Faye et al. (Eds.): CNRIA 2021, LNICST 400, pp. 40–51, 2021.
https://doi.org/10.1007/978-3-030-90556-9_4

network, ...) or by studying state of certain properties of the system (integrity of programs or stored data, user privileges, rights transfers, ...). How to detect different network attacks, especially ones that have never been seen before, is a key question researchers are trying to solve. To this end, to be able to assess the possible limits of current existing intrusion detection systems and to consider the contribution of Machine Learning techniques to improve intrusion detection systems, their intrusion detection process and also efficiently manage alerts, is a major concern in the field of computer security.

The rest of this paper is structured as follows: Sect. 2 deals with related works. Section 3 presents our Approach and architecture related vulnerabilities and the classification of machine learning techniques used to predict the type of vulnerability. Section 4 illustrates the description of our dataset used, the experimental set-up and our learning algorithm. Section 5 provide discussions on results of our analysis and make comparison with other approaches. We conclude this work in the Sect. 6.

2 Related Works

Intrusion Detection System (IDS) research, particularly that involving machine learning methods, is of great interest to computer security researchers. These authors deal with various machine learning methods (supervised, unsupervised, etc.). As data issues are very important for machine learning, several researchers deal with them. These authors discuss the quantity, quality, availability and compatibility with different machine learning methods.

In [1], Alex Shenfield et al. present an approach to detect malicious attacks traffic on networks using artificial neural networks based on deep packet inspection of the network's packets. The KDD CUP 1999 dataset was used for training. They achieved an average accuracy of 98% and an average false positive rate of less than 2%. This shows that the proposed classification technique is quite robust, accurate and precise, but the false alarm rate is still high.

N. Chockwanich et al. [2] used Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) to classify five types of attack using Keras with TensorFlow. To evaluate performance, they used the MAWI dataset which are pcap ("packet capture") files and compared their results with those of Snort. The results show that Snort could not detect the network scan attack via ICMP and UDP. They proved that RNN and CNN can be used to classify Port scan, Network scan via ICMP, Network scan via UDP, Network scan via TCP, and DoS attack with high accuracy. RNN offers an accuracy of 99.76% and 99.56% for CNN.

In [3] Leila Mohammadpour et al. used CNN and the NSL-KDD dataset for network intrusion detection system. They obtained, for their experimental results, a detection rate of 99.79%.

Peng Lin et al. [4] use Long-Short Term Memory (LSTM) to build a deep neural network model and add an attention mechanism (AM) to improve the model performance. They use the CSE-CIC-IDS2018 data set. Experimental

results show an accuracy of 96.2%, and they claim that this figure is better than the figures of other machine learning algorithms. Nevertheless the score is below that of other models using the same data set as shown in the work below.

In [5] V. Kanimozhi et al. compared the performance of several algorithms on the CSE-CIC-IDS2018 dataset. They showed that the artificial neural networks (ANN) performed significantly better than the other models with a score of 99.97% and an accuracy of 99.96%.

V. Kanimozhi et al. [6] propose a system which consists in detecting a botnet attack classification. The proposed system is created by applying an artificial neural network model to a CSE-CIC-IDS2018 data set. The proposed system offers a precision score of 99.97% and the average false positive rate is only 0.1%. However, the model is only applied to botnet intrusions and therefore does not have the capability to detect other types of attacks.

3 Methodology: Approach and Architecture

3.1 Approach

In order for Intrusion Detection Systems to perform well with an excellent false alarm rate and "zero-day" attack detection, they must be trained regularly with new data obtained from monitoring network traffic. More the IDS model is trained, more it will adjust to improve. Traditionally, training for artificial intelligence models takes place in one go ("One-Shoot") and then these models are used, whatever the duration, without learning new information. Different techniques including continuous learning and incremental learning [7,8] are introduced in an attempt to learn continuously. This has become a very fashionable field of research [7,9–11]. However, changing part of the parameter space immediately affects the model as a whole [12]. Another problem related to the progressive training of a Deep CNN model is the issue of catastrophic oblivion [13]. When a formed Deep CNN is exclusively recycled on new data, it results in the destruction of existing features learned from previous data.

But based on an algorithm proposed by Deboleena Roy et al. [14] we can exploit the advantages of incremental learning. In their work, to avoid the problem of catastrophic forgetting, and to keep the functionalities learned in the previous task, Deboleena Roy et al. propose a network composed of CNN that develops hierarchically as that new classes are being introduced. CNN are able to extract high-level features that best represent the abstract form of low-level features of network traffic connections. We chose to use CNNs because of the many advantages they offer, particularly their ability to select the most significant features themselves and their architecture gives them the ability to prioritize the selected features from the simplest to the most sophisticated.

3.2 Architecture

Inspired by hierarchical classifiers, the model we propose, Tree-CNN is composed of several nodes linked together to have a tree structure. The first node of the

tree structure is the "root" node where the first classification takes place. All the other nodes of the tree structure, except the "leaf" nodes, have a Deep Convolutional Neuronal Network (Deep CNN) which is trained to classify the entry for their "child" nodes. The classification process starts at the "root" node and the data is then passed on to the next "child" node, depending on the result of the classification of the "root" node. This node then classifies the data at its own level and transmits it in turn, depending of the result of the classification, to one of its "child" nodes. This process is repeated until a "leaf" node is reached. This is the end of the classification steps.

The Fig. 1 shows a three-level architecture of our model. The method of training an artificial neural network with such an architecture is described by the Algorithm 1. One of the advantages of the tree structure is that it allows a considerable reduction of the decision (prediction) time. The principle is to first detect whether or not it is an attack and then to classify the intrusion. This is done by a structure in the form of a tree as shown in Fig. 2. There are three levels:

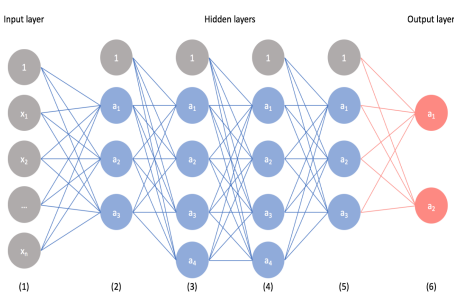

Fig. 1. Structure of a CNN

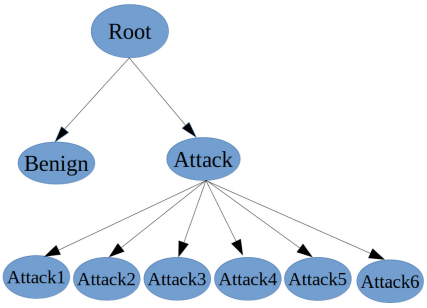

Fig. 2. Illustration of tree hierarchie

- the first level is the "root" node;
- the second level is made up of two nodes: a "leaf" node linked to the Benign class and a node linked to the Attack class;
- the third level contains "leaf" nodes, all from the "Attack" node.

All nodes, except the "leaf" nodes, contain a Deep CNN which is trained to classify the classes of its "child" nodes, each at its own level. The first node which is the root node is used to make the decision (to predict) whether it is an attack or a benign event. And if it's an attack, to give more details about the attack so that the administrator can intervene effectively, the node linked to the "Attack" class is led to make a classification. The node linked to the Attack class classifies the different types of attacks. The fourteen types of attacks, grouped into seven, are classified by this node.

4 Implementation

4.1 Tools Used

TensorFlow.[1] It is an open-source Google platform dedicated to machine learning. It offers a complete and flexible ecosystem of tools, libraries and of community resources to enable researchers to advance in the machine learning domain, and for developers to create and deploy applications that exploit this technology.

Pandas.[2] It is an open source library used for data manipulation and processing. It is a fast, powerful, flexible and easy to use data manipulation and analysis tool. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Keras.[3] It is an API for deep neural networks and is written in Python, and now included in TensorFlow. It focuses on ergonomics, modularity and extensibility. It was born within the framework of the ONEIROS project (Open-ended Neuro-Electronic Intelligent Robot Operating System). It was originally written by François Chollet in his book Deep learning with Python.

Scikit-Learn.[4] It is a free Python library for the learning machine. It has simple and effective tools for predictive data analysis. Built on NumPy, SciPy, and matplotlib, it is accessible and reusable in various contexts. It is open source, and is developed by numerous contributors, particularly in the academic world.

NumPy.[5] It is a library written in the Python programming language, designed to manipulate matrices or multidimensional arrays as well as mathematical functions operating on these arrays. It is an open source project aimed at enabling numerical computation with Python. It has been created in 2005, on the basis of initial work from the Numerical and Numarray libraries.

4.2 Data-Set

The dataset used is CSE CIC-IDS2018[6]. It is one of the most recent, realistic and up-to-date public data sets and more complete in terms of the types of attacks they contain. The attacks that the CSE CIC-IDS2018 contains are topical and best reflect threats networks and information systems of our companies, which are currently facing nowadays. The CSE CIC-IDS2018 dataset contains 86 "features". The data contained in CSE CIC-IDS2018 are labelled with fourteen labels which are presented in Table 2. The number of lines in the data set is over sixteen million (16,000,000) and is distributed as shown in Table 1 according to the type of attack.

[1] https://tensorflow.org.
[2] https://pandas.pydata.org.
[3] https://keras.io.
[4] https://scikit-learn.org.
[5] https://numpy.org.
[6] http://www.unb.ca/cic/datasets/ids-2018.html.

Table 2 clearly show that there is an imbalance in the data. The label "benign" alone represents more than 80% of the data. Indeed, in everyday use, benign cases far outweigh the cases of attack. But the model risks giving more weight to the benign label compared to others during training. This problem can be alleviated by using the SMOTE (Synthetic Minority Oversampling Technique) [15] to try out to balance the data. SMOTE is an over-sampling method, it works by creating synthetic samples from the class or the minority classes instead of creating simple copies.

Table 1. Distribution of CSE CIC-IDS2018 data

Label	Number
Benign	13484708
DDOS attack-HOIC	686012
DDoS attacks-LOIC-HTTP	576191
DoS attacks-Hulk	461912
Bot	286191
FTP-BruteForce	193360
SSH-Bruteforce	187589
Infilteration	161934
DoS attacks-SlowHTTPTest	139890
DoS attacks-GoldenEye	41508
DoS attacks-Slowloris	10990
DDOS attack-LOIC-UDP	1730
Brute force-Web	611
Brute force-XSS	230
SQL injection	87

Table 2. Breakdown of CSE CIC-IDS2018 data into "Benin" and "Attack"

Label	Number	Percentage
Benign	13484708	0,8307
Attack	2748235	0,1693

4.3 Learning Algorithm

We used Deboleena Roy et al.'s algorithm [14]. To train the model to recognize a number of new classes of attacks, we provide data on these attacks at the root node. We obtain a three-dimensional matrix at the output layer: $O^{K\times M\times I}$, where

K is the number (in our case 02) of child nodes of the root node,
M is the number of new classes of attacks and
I is the number of data samples per class.
$O(k;m;i)$ indicates the output of the k^{ith} node for the i^{ith} given belonging to the m^{ith} class, where $k \in [1,K]$, $m \in [1,M]$ and $i \in [1,I]$.
$O_{avg}(k,m)$ (Eq. (1)) is the average of the outputs(of the k^{ith} node and the m^{ith} class) on I data.
$O_{avg}^{K\times M}$ is the matrix of these averages over the I data.

The *Softmax* (Eq. (2)) is calculated on each average O_{avg} (Eq. (1)) to get a matrix $L^{K\times M}$.
An ordered list S is generated from the matrix $L^{K\times M}$, having the following properties:

- The list S has M objects. Each object is uniquely linked to one of the new classes M.
- Each object $S[i]$ has the following attributes:
  1. $S[i].label$ = label of the new class.
  2. $S[i].value = [v_1, v_2, v_3]$, the 3 highest *Softmax* values of the averages (O_{avg}) of this new class, ranked in descending order $v_1 \geq v_2 \geq v_3$.
  3. $S[i].nodes = [n_1, n_2, n_3]$, the nodes corresponding respectively to the values *Softmax* v_1, v_2, v_3.
- S is ordered by descending value of $S[i].value[1]$

$$O_{avg}\,(k,m) = \sum_{i=1}^{I} \frac{O\,(k,m,i)}{I} \tag{1}$$

$$L(k,m) = \frac{e^{O_{avg}(k,m)}}{\sum_{k=1}^{K} e^{O_{avg}(k,m)}} \tag{2}$$

This scheduling is done to ensure that new classes with high Softmax values are first added to the Tree-CNN tree. After building S, we examine its first element and take one of the three paths:

i. add the new class to an existing child node: If v_1 is greater than the next value (v_2) of a threshold, α (in our case $\alpha = 0,1$), this indicates a strong resemblance to the node n_1. The new class is then added this node;
ii. merge 2 child nodes and add the new class: If two of the values *Softmax* are close, i.e., when $v_1 - v_2 < \alpha$, and $v_2 - v_3 > \beta$ (a user-defined threshold, here we have defined it $\alpha = 0,1$), then, if n_2 is a leaf node, we merge n_2 into n_1 and add the new class to n_1;
iii. add the new class as a new child node: If the three values *Softmax* are not different with wide margin ($v_1 - v_2 < \alpha, v_2 - v_3 < \beta$: for example if the three largest values *Softmax* are $v_1 = 0,35$, $v_2 = 0,33$, and $v_3 = 0,31$), or if all child nodes are full, the network adding the new class as a new child node.

To prevent the Tree-CNN tree from becoming unbalanced, the maximum number of children that a branch node can have can be set. The procedure described above is repeated iteratively until all new classes are assigned a location below the root node. The pseudo code is described in the paper [14]. We also illustrate an example of incremental learning in Tree-CNN with the Fig 2.

5 Tests and Results

5.1 Execution Environment

The tests were carried out in a Cloud environment with shared resources. Indeed we used Google Colab in which we have access to 25.5 Gb of RAM memory, a GPU (Tesla P100-PCIE-16 GB) and a CPU (Intel(R) Xeon(R) CPU @ 2.30 GHz). But Google does not give any guarantee on the total availability of the promised resources. Our model is composed of ten (10) layers of convolutions interspersed with two (02) layers of Maxpooling, one Flatten layer, two (02) dense layers. All the convolution and dense layers, except the last one, have a *ReLU* activation. The last layer has *Sofmax* activation.

The training has been carried out over 2000 epochs. The Algorithm 1 shows how is the inference of our model.

5.2 Score and Other Measures

Four (04) basic elements are used to assess performance IDS. They can be represented in the form of a cross table Table 3. The actual metrics are:

The metrics derived from these basic elements are:

Algorithm 1: Inference algorithm [14]

```
I = Input Image, node = Root Node of the Tree
procedure ClassPredict(I, node)
    count = # of children of node
    if count = 0 then
        label = class label of the node
        return label
    else
        nextNode = EvaluateNode(I, node)
        ► returns the address of the child node of highest output neuron
        return ClassPredict(I, nextNode)
    end if
end procedure
```

Table 3. Basic elements for measuring the performance of an IDS

	Positive prediction	Negative prediction
Intrusions	TP (True Positive): intrusions are identified as intrusions	FN (False Negative): intrusions are identified as benigns
Benigns	FP (False Positive): benigns are identified as intrusions	TN (True Negative): benigns are identified as benigns

- **Accuracy**: it is defined as the ratio of correctly predicted samples to the total number of samples. The score is an appropriate measure only when the data set is balanced.

$$\textbf{Accuracy}: = \frac{TP+TN}{TP+FN+FP+TN}$$

- **Precision**: it is defined as the ratio of correctly predicted positive samples to predicted positive samples. It represents confidence in the detection of attacks.

$$\textbf{Precision}: = \frac{TP}{TP+FP}$$

- **Recall**: it is defined as the ratio of correctly predicted positive samples to the total number of actually positive samples. It reflects the system's ability to recognize attacks.

$$\textbf{Recall}: = \frac{TP}{TP+FN}$$

- **F1-measure**: It is defined as the harmonic mean of Precision and Recall. The higher rate of F1-measure shows that the system is performed better

$$\textbf{F1-measure}: = 2 \times \left[\frac{Precision \times Recall}{Precision+Recall}\right] = \frac{2 \times TP^2}{TP^2+TP \times (FN+FP)}$$

- **Rate of False Negative (RFN)**: it is defined as the ratio of intrusions are identified as benigns to the total number of intrusions. The RFN is also termed the Missed Alarm Rate.

$$RFN = \frac{FN}{TP+FN}$$

- **Rate of False Positive (RFP)**: it is defined as the ratio of benigns are identified as intrusions to the total number of benign. The RFP is also termed the False Alarm Rate.

$$TFP = \frac{FP}{VN+FP}$$

The tests for detection (performed by the "root" node) give the following results Table 4:

Table 4. Results of detection test

Accuracy	Precision	F1-measure	Recall	RFP	RFN
99,94%	99%	99%	99%	0.0001	0.0001

For the multi-class classification (performed by the "Attack" node), we obtain a accuracy of *97.54%*. The following Table 5 gives details.

Table 5. Results classification test

Classe	Precision	F1-measure	Recall
DDOS	100%	100%	100%
DoS	96,50%	92,71%	89,21%
Bot	100%	99,82%	99,64%
BruteForce	90,00%	93,43%	97,12%
Infilteration	97,19%	98,40%	99,64%
Web attack and injection	99,63%	98,73%	97,84%

5.3 Comparison with Related Works

It is difficult to compare work that does not use the same methods or dataset. So we have chosen to compare with work that uses similar machine learning techniques to our own. We have through the Table 6 summarized the results obtained in other works, particularly those using deep learning. In the cited works, some of them deal with intrusion classification (multi-classes) Table 7 and others deal with intrusion detection Table 6 (binary classification).

As far as detection is concerned, we obtained a score of 99.94%, an accuracy of 99%. These measurements are slightly lower than those of Kanimozhi et al. [6] (the best) cited in Table 6. But this can be explained by the fact that our model takes more parameters into account since, in addition to detection, it makes the classification, contrary to that of Kanimozhi et al. Also through the Table 6, we see very clearly that our model is better for multi-class classification. Indeed, the score largely exceeds those of the multi-class classification works cited in Table 6. The accuracy of our model on each class shows that it is not very wrong on classification. From all the above, we can say that our method is a good means of detecting and classifying intrusions.

Table 6. Table of detection performance of some models

Works	Methods	Dataset	Accuracy	Precision	F1-measure
T. Le et al. [16]	LSTM:binary	KDD Cup99	97.54	0.97	–
Zeng et al. [17]	1D-CNN:binary	ISCX2012	99.85	–	–
Zeng et al. [17]	LSTM:binary	ISCX2012	99.41	–	–
Kanimozhi et al. [6]	ANN:binary	CSE-CIC-IDS 2018	99,97	1,0	1,0
Our method	**Tree-CNN:binary**	**CSE-CIC-IDS 2018**	**99,94**	**0.99**	**0.99**

Table 7. Table of classification performance of some models

Works	Methods	Dataset	Accuracy	Precision	F1-measure
Q. Niyaz et al. [18]	RNN:5-class	NSL-KDD	79.10	–	–
S. Potluri et al. [19]	CNN:5-class	NSL-KDD	91.14	–	–
D. Yalei et al. [20]	CNN:5-class	NSL-KDD	80,13	–	–
S. Potluri et al. [21]	DBN+SVM: 5-class	NSL-KDD	92,06	–	–
Our method	**Tree-CNN: 6-class**	**CSE-CIC-IDS 2018**	**97,53**		

6 Conclusion

We proposed in this paper a model of Intrusion Detection System using CSE-CIC-IDS-2018 and Tree-CNN, a hierarchical Deep Convolutional Neural Network for incremental learning. We evaluated the performance of the model by comparing it with other approaches. Through this evaluation, we found that our model is more effective on multi-class classification than the others mentioned in the related works. Our approach therefore allows the classification and detection of intrusions with good performance. Nevertheless, there is still room for improvement. Research carried out in this paper can be further investigated in order to improve our approach and methods used. In terms of perspectives, this means:

- improve the model: there is still room for improvement in performance, for the multi-class classification of the model by further adjusting the hyper-parameters. In addition, the model can be trained with other recent datasets in order to reduce its false alarm rates and make it even more robust and reliable;
- from IDS to IPS: another way to improve the proposed solution would be to combine our model with rule-based intrusion detection tools such as Snort, which could speed up the detection of trivial or recurring cases. In addition, it would be interesting to further develop our solution so that it can trigger actions based on detected intrusions, i.e. it can act against an attack while waiting for the administrator's intervention. The final solution would no longer be an Intrusion Detection System (IDS), but an Intrusion Prevention System (IPS).

References

1. Shenfield, A., Day, D., Ayesh, A.: Intelligent intrusion detection systems using artificial neural networks. ICT Express **4**(2), 95–99 (2018). SI on Artificial Intelligence and Machine Learning
2. Chockwanich, N., Visoottiviseth, V.: Intrusion detection by deep learning with TensorFlow. In: 2019 21st International Conference on Advanced Communication Technology (ICACT), pp. 654–659 (2019)
3. Mohammadpour, C.S.L.L., Ling, T.C., Chong, C.Y.: A convolutional neural network for network intrusion detection system. In: Asia-Pacific Advanced Network (APAN) (2018)
4. Lin, P., Ye, K., Xu, C.-Z.: Dynamic network anomaly detection system by using deep learning techniques. In: Da Silva, D., Wang, Q., Zhang, L.-J. (eds.) CLOUD 2019. LNCS, vol. 11513, pp. 161–176. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23502-4_12
5. Kanimozhi, V., Prem Jacob, T.: Calibration of various optimized machine learning classifiers in network intrusion detection system on the realistic cyber dataset CSE-CIC-IDS 2018 using cloud computing. Int. J. Eng. Appl. Sci. Technol. **4**, 2455–2143 (2019)

6. Kanimozhi, V., Jacob, T.P.: Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS 2018 using cloud computing. In: 2019 International Conference on Communication and Signal Processing (ICCSP), pp. 0033–0036 (2019)
7. Giraud-Carrier, C.: A note on the utility of incremental learning. AI Commun. **13**(4), 215–223 (2000)
8. Ring, M.B.: Child: a first step towards continual learning. In: Thrun, S., Pratt, L. (eds.) Learning to Learn, pp. 261–292. Springer, Boston (1998). https://doi.org/10.1007/978-1-4615-5529-2_11
9. Polikar, R., Upda, L., Upda, S.S., Honavar, V.: Learn++: an incremental learning algorithm for supervised neural networks. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **31**(4), 497–508 (2001)
10. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: Advances in Neural Information Processing Systems, pp. 2990–2999 (2017)
11. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. Proc. Mach. Learn. Res. **70**, 3987 (2017)
12. XIao, T., Zhang, J., Yang, K., et al.: Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 177–186 (2014)
13. Goodfellow, I.J., Mirza, M., Xiao, D., et al.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211 (2013)
14. Roy, D., Panda, P., Roy, K.: Tree-CNN: a hierarchical deep convolutional neural network for incremental learning. Neural Netw. **121**, 148–160 (2020)
15. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
16. Le, T., Kim, J., Kim, H.: An effective intrusion detection classifier using long short-term memory with gradient descent optimization. In: 2017 International Conference on Platform Technology and Service (PlatCon), pp. 1–6 (2017)
17. Zeng, Y., Gu, H., Wei, W., Guo, Y.: Deep-full-range: a deep learning based network encrypted traffic classification and intrusion detection framework. IEEE Access **7**, 45182–45190 (2019)
18. Javaid, A., Niyaz, Q., Sun, W., Alam, M.: A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (Formerly BIONETICS), pp. 21–26 (2016)
19. Potluri, S., Ahmed, S., Diedrich, C.: Convolutional neural networks for multi-class intrusion detection system. In: Groza, A., Prasath, R. (eds.) MIKE 2018. LNCS (LNAI), vol. 11308, pp. 225–238. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05918-7_20
20. Ding, Y., Zhai, Y.: Intrusion detection system for NSL-KDD dataset using convolutional neural networks. In: Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence, pp. 81–85 (2018)
21. Potluri, S., Henry, N.F., Diedrich, C.: Evaluation of hybrid deep learning techniques for ensuring security in networked control systems. In: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–8 (2017)