2019 IEEE Symposium Series on Computational Intelligence (SSCI)
December 6-9 2019, Xiamen, China

LuNet: A Deep Neural Network for Network Intrusion Detection

Peilun Wu and Hui Guo
The University of New South Wales, Sydney, Australia
Email: {z5100023, huig}@cse.unsw.edu.au

***Abstract*—Network attack is a significant security issue for modern society. From small mobile devices to large cloud platforms, almost all computing products, used in our daily life, are networked and potentially under the threat of network intrusion. With the fast-growing network users, network intrusions become more and more frequent, volatile and advanced. Being able to capture intrusions in time for such a large scale network is critical and very challenging. To this end, the machine learning (or AI) based network intrusion detection (NID), due to its intelligent capability, has drawn increasing attention in recent years. Compared to the traditional signature-based approaches, the AI-based solutions are more capable of detecting variants of advanced network attacks. However, the high detection rate achieved by the existing designs is usually accompanied by a high rate of false alarms, which may significantly discount the overall effectiveness of the intrusion detection system. In this paper, we consider the existence of spatial and temporal features in the network traffic data and propose a hierarchical CNN+RNN neural network, LuNet. In LuNet, the convolutional neural network (CNN) and the recurrent neural network (RNN) learn input traffic data in sync with a gradually increasing granularity such that both spatial and temporal features of the data can be effectively extracted. Our experiments on two network traffic datasets show that compared to the state-of-the-art network intrusion detection techniques, LuNet not only offers a high level of detection capability but also has a much low rate of false positive-alarm.**

Keywords*—network intrusion detection; convolutional neural network; LuNet; recurrent neural network.

I. Introduction

Networked computing becomes indispensable to people's life. From daily communications to commercial transactions, from small businesses to large enterprises, all activities can be or will soon be done through networked services. Any vulnerabilities in the networked devices and computing platforms can expose the whole network under various attacks and may bring about disastrous consequences. Hence, effective network intrusion detection (NID) solutions are ultimately essential to the modern society.

Initial NID designs are signature-based, where each type of attacks should be manually studied beforehand, and the detection is performed based on the attack's signatures. This kind of approaches are, however, not suitable to the fast-growing network and cannot cope with attacks of increasing volume, complexity and volatility.

For the security of such a large scale and ever-expanding network, we need an intrusion detection system that is not only able to quickly and correctly identify known attacks but also adaptive and intelligent enough for the unknown and evolved attacks, which leads to AI-based solutions. The artificial intelligence gained from machine learning enables the detection system to discover network attacks without much need for human intervention [1].

So far, investigations on the AI-based solutions are mainly based on two schemes: anomaly detection and misuse detection. The anomaly detection identifies an attack based on its anomalies deviated from the profile of normal traffic. Nevertheless, this scheme may have a high false positive rate if the normal traffic is not well profiled, and the profile used in the detection is not fully representative [2]. Furthermore, to obtain a fully representative normal traffic profile for a dynamically evolving and expanding network is unlikely possible.

The misuse detection, on the other hand, focuses on the abnormal behaviour directly. The scheme can learn features of attacks based on a labelled training dataset, where both normal and attacked traffic data are marked. Given sufficient labelled data, a misuse-detection design can effectively generate an abstract hypothesis of the boundary between normal and malicious traffic. The hypothesis can then be used to detect attacks for future unknown traffic. Therefore the misuse detection is more feasible and effective than the anomaly detection [2] - [3] and has been adopted in real-world systems, such as Google, Qihoo 360 and Aliyun.

However, the existing misuse detection designs still present a high false positive rate, which significantly limits the in-time detection efficiency, incurs large manual scrutiny workload, and potentially degrades the network-wide security. In this paper, we address this issue, and we aim for an improved misuse detection design. Our contributions are summarized as follows:

- We present a hierarchical deep neural network, LuNet, that is made of multiple levels of combined convolution and recurrent sub-nets; At each level, the input data are learned by both CNN and RNN nets; As the learning progresses from the first level to the last level, the learning granularity becomes increasingly detailed. With such an arrangement, the synergy of CNN and RNN can be effectively exploited for both spatial and temporal feature extractions.
- We provide an in-depth analysis and discussion for the configuration of LuNet so that a high learning efficiency can be achieved.

978-1-7281-2485-8/19/$31.00 ©2019 IEEE

We test our design on two network intrusion datasets, NSL-KDD and UNSW-NB15, and we demonstrate that our design offers a higher detection capability (namely, better detection rate and validation accuracy) while maintaining a significantly lower false positive rate when compared to a set of state-of-the-art machine learning based designs.

The remainder of this paper is organized as follows: Section II provides a brief background of machine learning for network intrusion detection. The design of LuNet is detailed in Section III. The threat model in the datasets used for LuNet design is presented in Section IV. The evaluation and comparison of LuNet with other state-of-the-art designs are given in Section V. The paper is concluded in Section VI.

II. Background and Related Work

Many algorithms have been developed for machine learning. They can be classified into classical machine learning approaches and deep learning approaches. Some of them, relevant to network intrusion detection are briefly discussed below.

A. Classical Machine Learning Approaches

Among many machine learning approaches [4], kernel machines and ensemble classifiers are two effective schemes that can be considered for NID. Support Vector Machine (SVM) that has long been used for task classification is a typical example of kernel machines, and Radial Basis Function (RBF) kernel (also called Gaussian kernel) is the most used kernel function [5]. SVM makes the data that cannot be linearly separated in the original space, separable by projecting the data to a higher-dimension feature space. Ensemble classifiers, such as Adaptive Boosting [6], Random Forest [7], are often constructed with multiple weak classifiers to avoid overfitting during training so that a more robust classification function can be achieved.

However, both kernel machines and ensemble classifiers are not scalable to a large data set. Their validation accuracy rarely scales with the size of training data [8]. Given a high volume of training data available from the large scale network, using these traditional machine learning approaches to train the intrusion detection system is not efficient. Furthermore, the traditional approaches learn input data only based on a given set of features; they cannot generalize features from raw data, and their learning efficiency highly relies on the features specified.

B. Deep Learning Approaches

Deep learning organizes "learning algorithms" in layers in the form of "artificial neural network" that can learn and make intelligent decisions on its own.

Multi-Layer Perceptron (MLP) [9] is an early class of feed-forward deep neural network that utilizes the backpropagation algorithm to minimize the error rate during training. MLP was initially used to solve complex approximate problems for speech recognition, image recognition, and machine translation.

The real flourish and practical breakthrough of deep learning stems from two popular deep learning algorithms: the convolutional neural network (CNN) and recurrent neural network (RNN). CNN can automatically extract features of raw data and has gained great successes for image recognition [10] - [11] - [12]. However, the feature map generated by CNN often manifests the spatial relations of data. CNN does not work very well for the data of long-range dependency.

RNN, on the other hand, has an ability to extract the temporal features from the input data. Long short-term memory (LSTM) is a popular RNN [13]. It keeps a trend of the long-term relationship in the sequential data while being able to drop out short-lived temporal noises.

Recently, a hierarchical convolutional and recurrent neural network, HAST-IDS [14] has been used to learn spatial and temporal features from the network traffic data.

The LuNet proposed in this paper is also a hierarchical network. Our design is similar to HAST-IDS in that both utilize CNN and RNN for spatial-temporal feature extraction. However, there are some major differences:

1) HAST-IDS stacks all RNN layers after CNN layers, while LuNet has a hierarchy of combined CNN and RNN layers.
2) In HAST-IDS, because the CNN hierarchy is placed before RNN, the deep CNN may drop out the temporal information embedded in the raw input data, which makes RNN learning ineffective. LuNet, on the other hand, synchronizes both the CNN learning and the RNN learning into multiple steps with the learning granularity being gradually increased from coarse to fine; Therefore, both spatial and temporal features can be adequately captured.
3) We apply batch normalization between CNN and RNN so that the learning efficiency and accuracy can be further improved.

The design of LuNet is presented in the next section.

III. LuNet

A. Overview Structure

As stated above, CNN targets on spatial features while RNN aims for temporal features. The existing design HAST-IDS simply structures CNN and RNN in tandem, as illustrated in Fig.1(a). When the learning progresses along the multiple levels in the CNN hierarchy, the information extracted becomes more spatial oriented. The temporal features may be lost by the CNN hierarchy, which significantly limits the learning effectiveness of the following RNN (LSTM).

Rather than allow CNN to learn to its full extent first, in LuNet, we mingle the CNN and RNN subnets and synchronize both the CNN learning and the RNN learning into multiple steps and each step is performed by a combined CNN and RNN block, or *LuNet block*, as illustrated in Fig.1(b). Since CNN is able to extract high-level features from a large amount of data, we place CNN before RNN at each level. The learning starts from the first step on coarse-grain learning, hence the

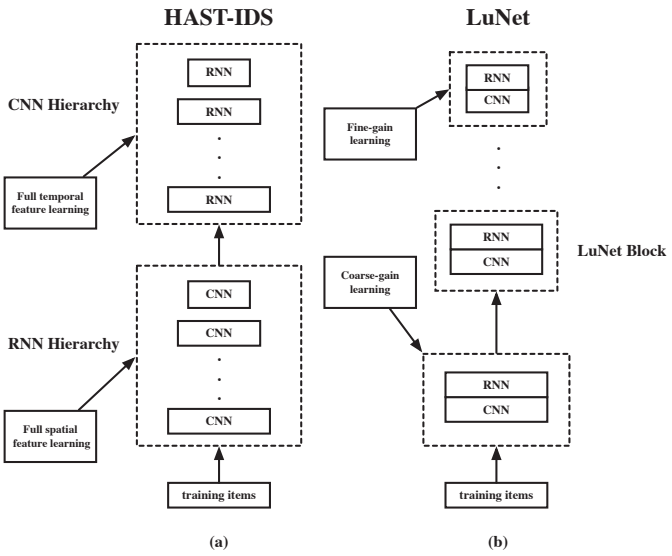


Fig. 1: Different CNN and RNN arrangements. (a) HAST-IDS, CNN and RNN in tandem (b) LuNet, CNN and RNN mingled.

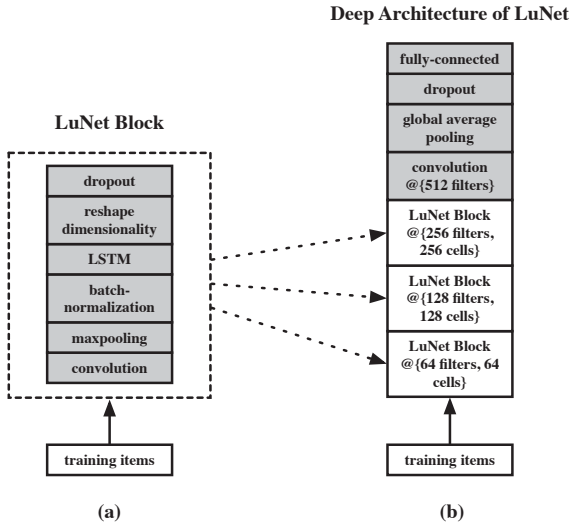


Fig. 2: LuNet. (a) a LuNet Block. (b) the overall deep architecture of LuNet.

CNN output will still retain temporal information that will then be captured by RNN. The learning granularity becomes detailed as the data processing flows to the next step; but at each level, both CNN and RNN learn input on the same granularity. In such a way, CNN and RNN can learn to their full capacity without much interference with each other.

Given the overall structure of LuNet, the effectiveness of its learning is closely related to the hidden layers design, which is discussed in the next sub section.

B. Hidden Layers Design

We measure the learning granularity in terms of the number of filters/cells used in the CNN/RNN network. The smaller the filter/cell number, the larger the granularity. Fig.2 shows a LuNet with three CNN+RNN levels. The filter/cell number increases from 64 for the first level to 256 for the last level.

The design considerations for each layer in a LuNet block and the final four processing layers for the learning outputs, as highlighted in shaded boxes in Fig.2, are given below.

1) Convolutional Neural Network (CNN): CNN mainly consists of two operations: convolution and pooling. Convolution transforms input data, through a set of filters or kernels, to an output that highlights the features of the input data, hence the output is usually called feature map. The convolution output is further processed by an activation function and then down-sampled by pooling to trim off irrelevant data. Pooling also helps to remove glitches in the data to improve the learning for the following layers [15] - [16].

CNN learns the input data by adjusting the filters automatically through rounds and rounds of learning processes so that its output feature map can effectively represent the raw input data.

Since the network packet is presented in a 1D format, we use 1D convolution[1] in LuNet, as illustrated in Formula (1) that specifies the operation to the input vector g with a filter f of size m.

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2), \tag{1}$$

where i is the position of different values in the sequence data.

Because the rectified linear unit (ReLU): $f(z) = max(0, z)$ is good for fast learning convergence, we therefore, choose it as the activation function. We also use the max pooling operation, as commonly applied in other existing designs [17].

2) Batch Normalization: One problem with using the deep neural network is that the input value range dynamical changes from layer to layer during training, which is also known as covariance shift. The covariance shift causes the learning efficiency of one layer dependent on other layers, making the learning outcome unstable. Furthermore, because of the covariance shift, the learning rate is likely restricted to a low value to ensure data in different input ranges to be effectively learned, which slows down the learning speed. Batch normalization can be used to address this issue.

Normalization scales data to the unit norm in the input layer and has been used to accelerate training deep neural network for image recognition [18].

We use batch normalization to adjust the CNN output for RNN in a LuNet block. The normalization subtracts the batch mean from each data and divides the result by the batch standard deviation, as given in Formula (2).

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\delta_B^2 + \epsilon}}, \tag{2}$$

[1]One issue involved in convolution multiplication is to maintain the kernel size as same as the input, for which two typical schemes can be used: zero or no-zero padding. Since both schemes do not generate much difference to the learning outcome, here we choose no-zero padding for simplicity.

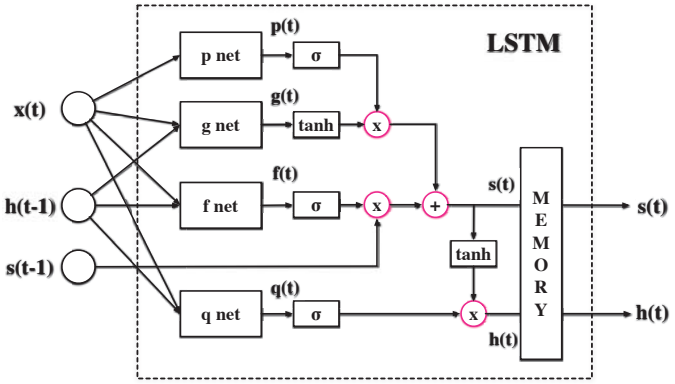


Fig. 3: A high level data processing diagram of LSTM.

where x is a value in the input batch, and μ_B and δ_B are, respectively, the batch mean and variance. The ϵ is an ignorable value, just to ensure the denominator in the formula non-zero.

Based on the normalized $\hat{x}$, the normalization produces the output y as given in Formula (3), where the γ and β will be trained in the learning process for a better learning outcome.

$$\hat{y} = \gamma\hat{x} + \beta. \tag{3}$$

3) Long Short Term Memory (LSTM): Different from CNN that learns information on an individual-data-record basis, RNN can establish the relationship between data records by feeding back what has been learned from the previous learning to the current learning, and hence can capture the temporal features in the input data.

However, the simple feedback used in the traditional RNN may have a learning error accumulated in the long dependency. The accumulated errors may become large enough to invalid the final learning outcome. LSTM (Long Short-Term Memory), a gated recurrent neural network, mitigates such a problem. It controls the feedback with a set of gate functions such that the short-lived errors are eventually dropped out and only persistent features are retained. Therefore, we use LSTM for RNN.

For a brief description of LSTM operation, we create a high-level data processing diagram of LSTM [2].

LSTM can be abstracted as a connection of four sub-networks (denoted as p-net, g-net, f-net and q-net in the diagram), a set of control gates, and a memory component. The input and output values in the diagram are vectors of the same size determined by the input $x(t)$. The state, $s(t)$, saved in the memory, serves as the feedback to the current learning.

All the sub nets in LSTM have a similar structure, as specified in Formula (4).

$$b + U \times x(t) + W \times h(t-1), \tag{4}$$

where $x(t)$, $h(t-1)$, b, U and W are, respectively, current input, previous output, bias, weight matrix for the current input, and recurrent weight matrix for the previous output. Each of the four nets have a different b, U, and W.

The outputs from the sub nets ($p(t)$, $g(t)$, $f(t)$ and $q(t)$) are then used, through two types of controlling gates (σ and $\tanh$) to determine the feedback s(t) from the previous learning and the current output h(t), as given in Formulas (5) and (6), respectively.

$$s(t) = \sigma(f(t)) * s(t-1) + \sigma(p(t)) * \tanh g(t), \tag{5}$$

$$h(t) = \tanh s(t) * \sigma(q(t)). \tag{6}$$

LSTM learns the inputs by adjusting the weights in those nets and the σ value such that the temporal features between the input data can be effectively generated in the output.

4) Dimension Reshape: Since in LuNet, the learning granularity changes from one CNN+RNN level to another, the output size of one level is different from the input size expected at the next level. We, therefore, add a layer to reshape the data for the next LuNet block.

5) Overfitting Prevention: One typical problem when learning big data using a deep neural network is overfitting – namely, the network has learned the training data too well, which restricts its ability to identify variants in new samples. This problem can be handled by Dropout [19]. Dropout randomly removes some connections from the deep neural network to reduce overfitting. We add the dropout layer with a default rate value of 0.5 after the CNN+RNN hierarchy in LuNet.

6) Final Layers: Finally, an extra convolution layer and a global average pooling layer are used to extract further spatial-temporal features learned through the LuNet blocks. The final learning output is generated by the last layer, a fully-connected layer.

IV. Datasets and Threat Model

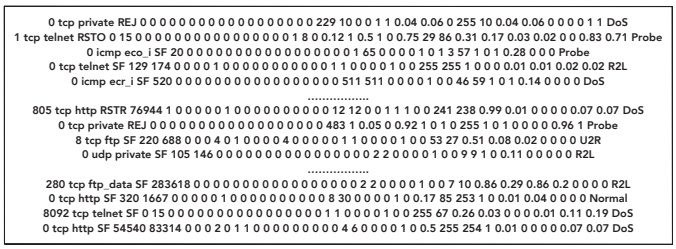
The evaluation of a neural network design is closely related to the dataset used. Many datasets collected for NID contain significant amount of redundant data [20] - [21], which makes evaluation results unreliable [5] - [6] - [7] - [14] - [22]. To ensure the effectiveness of the evaluation, we select two non-redundant datasets: NSL-KDD and UNSW-NB15 in our investigation.

NSL-KDD [23] was generated by *Canadian Institute for Cyber Security (CICS)* from the original KDD'99 dataset. It consists of 39 different types of attacks. The attacks are categorized into four groups: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L) and Probe. A big issue with this dataset is its imbalanced distribution, which often leads to a high false positive rate due to insufficient data available for training [24] - [25]. Here, we tackle this problem with a cross-validation scheme[3].

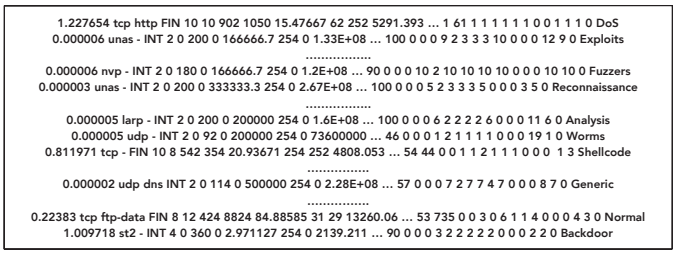
UNSW-NB15 [27] - [28], generated by *Australian Center for Cyber Security (ACCS)* in 2015, is a more contemporary dataset. For the dataset, the attack samples were first collected

[2]For the mathematical design details of LSTM, the reader is referred to the original paper [16] as shown in Fig. 3.

[3]This problem was also addressed in [26] with a different focus and strategy.



(a)



(b)

Fig. 4: Raw network traffic data. (a) NSL-KDD (41 features), (b) UNSW-NB15 (42 features).

from the three real-world websites: CVE (Common Vulnerabilities and Exposures)[4], BID (Symantec Corporation)[5], and MSD (Microsoft Security Bulletin)[6]. The sample attacks were then simulated in a laboratory environment for the dataset generation. There are nine attack categories in UNSW-NB15: DoS, Exploits, Generic, Shellcode, Reconnaissance, Backdoor, Worms, Analysis, and Fuzzers.

V. Evaluation and Discussion

To evaluate our design, we have implemented LuNet (as given in Fig. 2) with TensorFlow backend, Keras and scikit-learn packages and we run the training on a HP EliteDesk 800 G2 SFF Desktop with Intel (R) Core (TM) i5-6500 CPU @ 3.20 GHz processor and 16.0 GB RAM. For comparison, we also implemented a set of state-of-the-art machine learning algorithms.

The description of experiments and experiment results and discussion are presented below.

A. Data Preprocessing

1) Convert Categorical Features: For the experiment to be effective, we need data to conform to the input format required by the neural network. Raw network traffic data include some categorical features as shown in Fig.4. The text information cannot be processed by a learning algorithm and should be converted into numerical values. Here, we use the 'get_dummies' function in Pandas [29] to operate the conversion.

[4]CVE: https://cve.mitre.org/

[5]BID: https://www.securityfocus.com

[6]MSD: https://docs.microsoft.com/en-us/security-updates/securitybulletins

TABLE I: Result of Binary Classification

k	NSL-KDD			UNSW-NB15		
	DR%	ACC%	FPR%	DR%	ACC%	FPR%
2	98.55	99.09	0.41	98.50	96.65	6.60
4	98.56	99.11	0.37	98.04	97.51	3.44
6	99.18	99.30	0.59	98.12	97.62	3.26
8	99.27	99.34	0.59	97.78	97.67	2.54
10	99.00	99.36	0.7	98.45	97.57	3.96
average	99.42	99.24	0.53	98.18	97.40	3.96

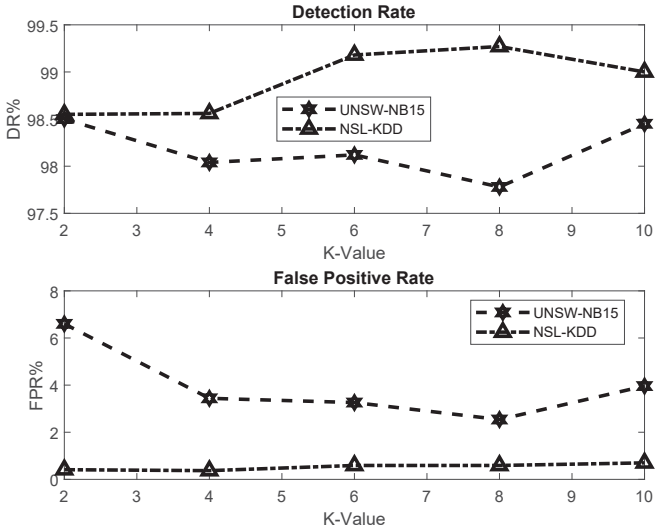


Fig. 5: A comparison of detection rate (DR%) and false positive rate (FPR%) on using LuNet for binary classification based on NSL-KDD and UNSW-NB15 datasets.

2) Standardization: Input data may have varied distributions with different means and standard derivations, which may affect learning efficiency. We apply standardization to scale the input data to have a mean of 0 and a standard deviation of 1, as is often applied in many machine learning classifiers.

3) Stratified K-Fold Cross Validation: NSL-KDD and UNSW-NB15 contain 148,516 and 257,673 samples, respectively. To realize the large non-redundant data for training and verification, we employ a Stratified K-Fold Cross Validation strategy, also commonly used in machine learning. The scheme splits all samples in a dataset into k groups; Among them, $k-1$ groups, as a whole, are used for training and the rest one group for validation; hence the strategy is also called *Leave One Out Strategy*.

B. Evaluation Metrics

We evaluate LuNet in terms of the validation accuracy (ACC), detection rate (DR) and false positive rate (FPR). ACC measures LuNet's ability to correctly predict both attacked and non-attacked normal traffic, while DR indicates its ability of prediction for attacks only. A high DR can be shadowed by a high rate of false positive alarm (FPR), which therefore needs

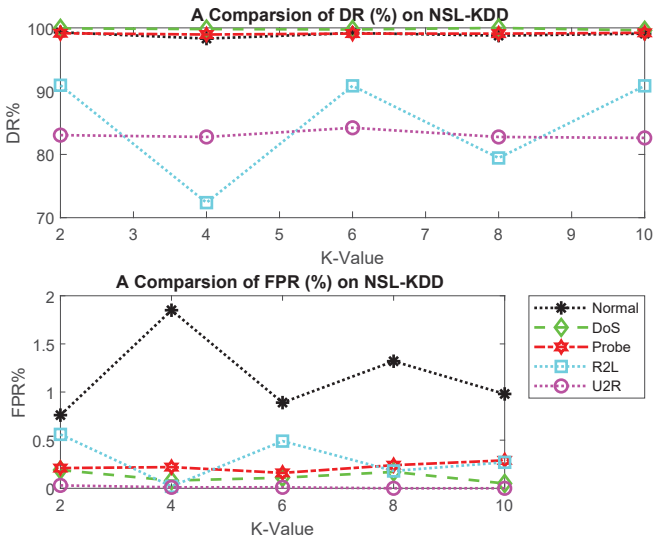


Fig. 6: A comparison of detection rate (DR%) and false positive rate (FPR%) on using LuNet for multi-class classification based on NSL-KDD.

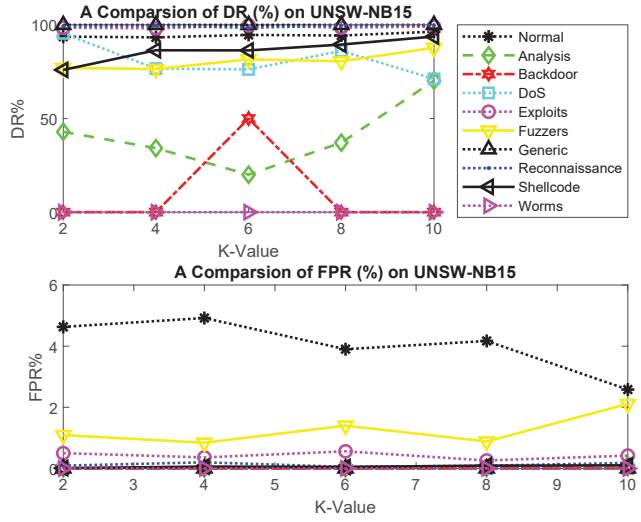


Fig. 7: A comparison of detection rate (DR%) and false positive rate (FPR%) on using LuNet for multi-class classification based on UNSW-NB15.

to be jointly considered with DR. The detail definitions of the three metrics are given in Formulas (7), (8) and (9).

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}, \tag{7}$$

$$DR = \frac{TP}{TP + FN}, \tag{8}$$

$$FPR = \frac{FP}{FP + TN}, \tag{9}$$

where TP and TN are, respectively, the number of attacks and the number of normal traffic correctly classified; FP is the number of actual normal records mis-classified as attacks, and FN is the number of attacks falsely classified as normal traffic.

TABLE II: RESULT OF MULTI-CLASS CLASSIFICATION

k	NSL-KDD			UNSW-NB15		
	DR%	ACC%	FPR%	DR%	ACC%	FPR%
2	99.24	99.09	0.98	95.37	84.61	1.69
4	98.15	98.88	0.33	95.08	84.78	1.45
6	99.11	99.13	0.77	96.10	84.93	2.09
8	98.68	99.02	0.58	95.83	85.21	1.32
10	99.02	99.14	0.61	97.43	85.35	2.89
average	98.84	99.05	0.65	95.96	84.98	1.89

C. LuNet Results

In our experiments, we use RMSprop [30] (a popular gradient descent algorithm) to optimize weights and biases for LuNet training, and for the learning rate, we set it to a median value 0.001 in a range given by the tensorflow library. We also choose the default rate value, 0.5, for the dropout layer.

We first measure the performance of LuNet based on two scenarios: (1) binary classification, namely LuNet predicts a packet either as an attack or as a normal traffic; (2) multi-class classification, where LuNet identifies a packet either as normal or as one type of attacks given in the dataset attack model (namely, 5 classes for NSL-KDD and 10 classes for UNSW-NB15). The experiment results are given below.

1) Binary Classification: Table I shows the detection rate, accuracy and false positive rate for the binary classification of LuNet under different Stratified K-Fold Cross Validations, with k ranging from 2 to 10. The average values are given in the last row of the table. As can be seen from the table, LuNet can achieve around 99.24% and 97.40% validation accuracy on NSL-KDD and UNSW-NB15, respectively. The best validation accuracy of 99.36% on the NSL-KDD dataset and 97.67% on the UNSW-NB15 can be observed. It can also be seen that LuNet offers a high detection capability while incurs a low false positive alarm rate, as plotted Fig.5. On average, *DR=99.42%* and *FPR=0.53%* for NSL-KDD, and *DR=98.18%* and *FPR=3.96%* on UNSW-NB15 can be obtained.

It is worth to note that because of the cross-validation used, the false positive alarm rate is not affected by the imbalanced NSL-KDD dataset, as is expected.

2) Multi-Class Classification: Table II shows the multi-class classification results. It can be seen from the table that LuNet can achieve an average of 99.05% accuracy, 98.84% detection rate on NSL-KDD, and 84.98% accuracy and 95.96% detection rate on UNSW-NB15 with, respectively, the 0.65% and 1.89% false positive rate on the two datasets.

Fig.6 shows the detection rate (DR%) and false positive rate (FPR%) for 5 category (Normal, DoS, Probe, R2L, U2R) classification on NSL-KDD. As we can see from the figure,

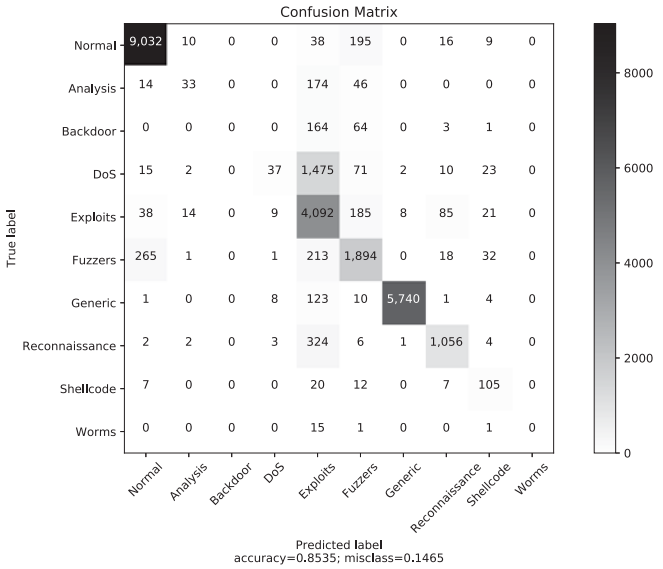


Fig. 8: The confusion matrix of using LuNet on the UNSW-NB15 data set when $k = 10$ for multi-class classification.

TABLE III: A COMPARISON OF 8 ALGORITHMS FOR MULTI-CLASS CLASSIFICATION BASED ON UNSW-NB15 DATASET WITH $K = 10$ FOR CROSS VALIDATION

Algorithm		DR%	ACC%	FPR%
ML	SVM (RBF)	83.71	74.80	7.73
	RF	92.24	84.59	3.01
	AdaBoost	91.13	73.19	22.11
	average	**89.03**	**77.53**	**10.95**
DL	CNN	92.28	82.13	3.84
	MLP	96.74	84.00	3.66
	LSTM	92.76	82.40	3.63
	HAST-IDS	93.65	80.03	9.60
	LuNet	97.43	85.35	2.89
	average	**94.57**	**82.78**	**4.72**

LuNet shows an excellent capability to handle attacks in all categories (with a high detection rate and low false positive rate), except for U2R and R2L. For U2R and R2L, LuNet presents a relatively low detection rate, which indicates that the main features extracted by LuNet for U2R and R2L are not effectively distinct. There may be significant feature overlap between U2R and other attacks. The same reason is also applied to the Backdoor and Worms attacks, as manifested in the similar plot shown in Fig.7 for classification of 10 categories (Normal, Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, Worms) on UNSW-NB15.

Fig.7 illustrates that LuNet can successfully detect most of the Normal traffic, Exploits, Generic, DoS, Shellcode and Reconnaissance attacks (see the overlapped lines at the very top of the plots) and has a moderate capability to detect Analysis attacks with the low false positive rate (FPR). However, it is unable to discover Backdoor and Worms attacks, as can be observed in the confusion matrix (as shown in Fig. 8) generated from the experiment. We discover that most of Backdoor and Worms were, in fact, classified as Exploits attacks by LuNet. The possible reason is that the Backdoor and Worms attacks use the common exploit flaws in the programs that listen for connections from remote hosts and the signature of those attacks is similar to the Exploits attacks. Therefore, the Backdoor and Worms attacks are treated as the Exploits attacks by LuNet. Another possible reason may come from the insufficient data available for the Backdoor and Worms attacks; There are only around 1.4% Backdoor attacks and 1.1% Worms attacks in the dataset.

D. Comparative Study

We compare the performance of LuNet with other five state-of-the-art supervised learning algorithms that have been investigated in the literature: Support Vector Machine (SVM) with Gaussian Kernel (RBF) [5], Multi-Layer Perceptron (MLP) [31], Random Forest (RF) [7], Adaptive Boosting (AdaBoost) [6], and HAST-IDS [14].

We also run the experiment on individual CNN and LSTM networks, presented in LuNet.

The detection rate (DR%), accuracy (ACC%), and false positive rate (FPR%) of the two design groups (ML, the classical machine learning algorithms and DL, the deep learning algorithms) are given in Table III. The average values of each group are also provided in the table.

As can be seen from Table III, the deep neural networks generally have better performance than the three classical machine learning algorithms; On average, the deep neural networks have a higher detection rate (94.57%) and accuracy (82.78%) and a lower false positive rate (4.72%), as compared to the corresponding average values of 89.03%, 77.53% and 10.95% from the classical machine learning. Among the five deep learning networks, LuNet is better than other four DL algorithms due to the special combined CNN+RNN hierarchy used. In summary, LuNet demonstrates its superiority – achieving a high detection rate and accuracy, while keeping a low false positive rate.

VI. CONCLUSION

In this paper, we present a deep neural network architecture, LuNet, to detect intrusions on a large scale network. LuNet uses CNN to learn spatial features in the traffic data and LSTM for temporal features. To avoid the information loss due to different learning focuses of CNN and RNN, we synchronize both CNN and RNN to learn the input data at the same granularity. To enhance the learning, we also incorporate batch normalization in the design. Our experiments on the two non-redundant datasets, NSL-KDD and UNSW-NB15, show that LuNet is able to effectively take advantages of CNN and LSTM. Compared with other state-of-the-art techniques, LuNet can significantly improve the validation accuracy and reduce the false positive rate for network intrusion detection.

It must be pointed out that LuNet does not work well to classify attacks of insufficient samples in the training dataset, such as Backdoors and Worms, as observed in our experiment, which will be investigated in the future.

REFERENCES

[1] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, 1994.

[2] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, pp. 305–316, IEEE, 2010.

[3] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[4] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[5] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018.

[6] W. Hu, J. Gao, Y. Wang, O. Wu, and S. Maybank, "Online adaboost-based parameterized methods for dynamic distributed network intrusion detection," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 66–82, 2013.

[7] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 5, pp. 649–659, 2008.

[8] Y. Bengio, O. Delalleau, and N. L. Roux, "The curse of highly variable functions for local kernel machines," in *Advances in Neural Information Processing Systems*, pp. 107–114, 2006.

[9] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.

[10] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[16] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*, vol. 1. Citeseer, 2017.

[17] Y.-T. Zhou and R. Chellappa, "Computation of optical flow using a neural network," in *IEEE International Conference on Neural Networks*, vol. 1998, pp. 71–78, 1988.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[20] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2, pp. 12–26, IEEE, 2000.

[21] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[22] X. Zhuo, J. Zhang, and S. W. Son, "Network intrusion detection using word embeddings," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 4686–4695, IEEE, 2017.

[23] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–6, IEEE, 2009.

[24] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *International Conference on Neural Information Processing*, pp. 858–866, Springer, 2017.

[25] S. Z. Lin, Y. Shi, and Z. Xue, "Character-level intrusion detection based on convolutional neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2018.

[26] P. Wu, H. Guo, and R. Buckland, "A transfer learning approach for network intrusion detection," in *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*, pp. 281–285, IEEE, 2019.

[27] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, IEEE, 2015.

[28] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.

[29] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[30] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Lecture Note*, vol. 14, p. 8, 2012.

[31] J. Esmaily, R. Moradinezhad, and J. Ghasemi, "Intrusion detection system based on multi-layer perceptron neural networks and decision tree," in *2015 7th Conference on Information and Knowledge Technology (IKT)*, pp. 1–5, IEEE, 2015.