Implementing an Intrusion Detection System Based on Convolutional Neural Network

Ali Abbas
University of Central Florida
Orlando, FL, US
ali.abbas@knights.ucf.edu

Serge Metellus
University of Central Florida
Orlando, FL, US
serge.metellus@knights.ucf.edu

ABSTRACT

The modern society heavily relies on internet and computer networks in areas such as commerce, education, and communication. Attacks on these networks cost billions of dollars each year. The traditional methods of intrusion detection in networks rely on hand crafted rules which require constant updates due to the rapid development of new types of attacks on networks. With the constant growth of networks and the huge amount of data carried by these networks, the traditional methods become increasingly impractical. Machine learning has been used to automate the detection of network attacks. It has the ability to learn patterns of intrusion autonomously and detect them in real time. The paper "An Intrusion Detection System Based on Convolutional Neural Network" by Pengju Liu is a recent example of using Convolutional Neural Network (CNN) in the field of network intrusion detection. CNN is a relatively new method in this area of research, and since reproducibility is one of the main goals of any scientific research, we reproduced the results of this paper independently and were able to confirm its validity.

CCS Concepts
• Security and privacy→Artificial immune systems;

Keywords
Intrusion detection system; LeNet-5; Convolutional neural network; CNN.

1. INTRODUCTION

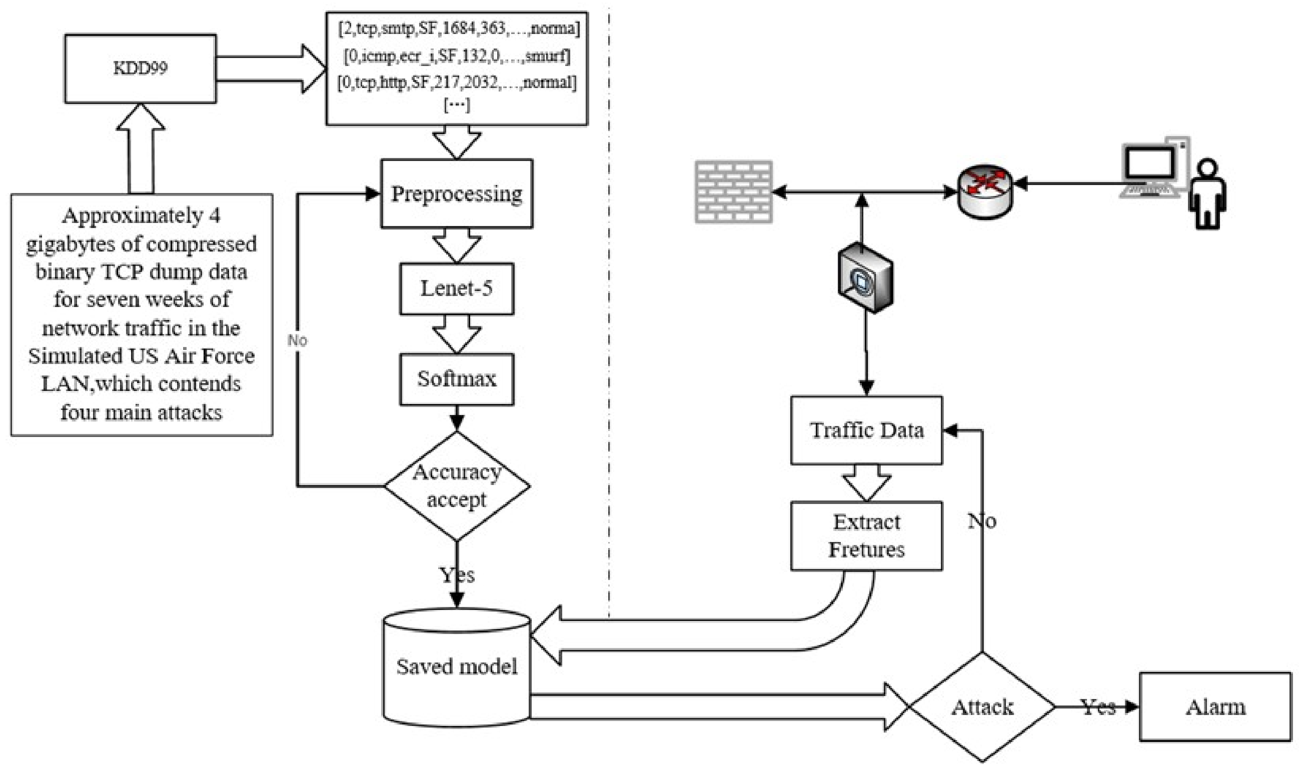
Attacks on internet and computer networks pose a serious security risk to many areas of modern life. Intrusion Detection System (IDS) detects these attacks by identifying dangerous events in the network or a host. Various traditional machine learning methods have been used for IDS but they struggle with huge amounts of data and multiple classification problem. Recently, Deep Learning (DL) methods such as Convolutional Neural Network (CNN) has emerged as powerful tools to deal with such huge amounts of data. Here we examine a recent paper "An Intrusion Detection System Based on Convolutional Neural Network" by Pengju Liu, that used CNN for IDS and tried to reproduce its results. Reproducibility is a major concern in scientific publications and it forms the foundation on which future studies can be built. If we can't reproduce the same results, there's little hope for using that result and trying to improve it.

The remainder of the paper is structured as follows. In section 2, the IDS system and its components are explained. Section 3 presents the results and discusses its implications. Finally, section 4 concludes the paper and suggests some directions for future work.

2. INTRUSION DETECTION SYSTEM WITH CNN

The Intrusion Detection System proposed by the paper [1] is composed of four main parts:

1) Data Acquisition: this part is responsible for collecting data for training the model. In this case we use the KDD99 dataset instead of this module.
2) Data preprocessing: prepares the data to be used by CNN.
3) CNN model: it uses the training data to detect the network attack patterns from the data.
4) Attack handling: It uses the data collected during the normal network operation in realtime and sends it to the trained CNN



KDD99
[2,tcp,smtp,SF,1684,363,…,norma]
[0,icmp,ecr_i,SF,132,0,…,smurf]
[0,tcp,http,SF,217,2032,…,normal]
[···]
Approximately 4 gigabytes of compressed binary TCP dump data for seven weeks of network traffic in the Simulated US Air Force LAN,which contends four main attacks
Preprocessing
Lenet-5
Softmax
No
Accuracy accept
Yes
Saved model
Traffic Data
Extract Fretures
No
Attack
Yes
Alarm

Figure 1. Intrusion Detection System with CNN (image from [1])

model. The CNN then detects if data is normal or an attack. In case of an attack the security measures are performed by the attack handling module.

Figure 1. Which is from [1] shows the structure of the Intrusion Detection System with CNN.

In Figure 1, the system on the left of the vertical dotted line creates the Intrusion Detection Model using CNN with training data. On the top right we see a real network system and a device to capture its traffic data. Then we extract features from traffic data and feed it to our model. If the model labels the data as an attack, alarm will be raised and security measures will follow to prevent the intrusion.

2.1 Data Preprocessing

In order to use CNN we need to first prepare the dataset and produce a format that can be accepted by CNN. Data preprocessing consists of three steps:

1. Convert Strings to Numbers: convert string values for categorical data to one-hot encoded numerical data (ex. [a,b,c] > [[0,0,1][0,1,0][1,0,0]])
2. Data Normalization: Because range of each data field is different from other data fields, we need to normalize them to range 0 to 1
3. Convert 1D records to 2D Matrix: Since CNN was originally meant to deal with images, it needs 2D input. We first pad the 1D record with zeros such that the square root of the length is a whole integer d, then reshape it to 2D matrix of size d * d. The author of paper [1] failed to mention this step but it's a very important step, since there are different ways of converting 1D to 2D, and could lead to different results.

2.2 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a type of deep neural networks that is mostly used in classifying images, but it has also been applied to a wide variety of classification tasks. It consists of multiple layers of convolution kernels which detect local features of the input, followed by subsampling layers to reduce the feature vector size (pooling) and finally a fully connected layer with Softmax output layer which outputs the classification assigned to the input. CNN consists of one or multiple layers of the following types:

- Convolution: To detect higher level features of the input one or more layers of convolution is used. A feature map m is obtained by convolving a kernel (filter) across input features and then activated by a nonlinear function. The nonlinear function that is often used, is a rectified linear unit (ReLU) which is defined as f(x) = Max(0, x).
- Pooling: To reduce and summarize the feature map we use average pooling, which selects the mean value in a sliding window.
- Fully connected: The 2D output of previous layers is flattened to 1D which becomes the input to this layer. The fully connected layers classifies the input and output the result in a form that is not readily accessible. We need to apply the softmax function to turn those numbers into class probabilities and finally into class labels.

Following the method used in the paper [1], we will use a LeNet-5 CNN. Its structure is shown in Figure 2.

The LeNet-5 CNN consists of these layers [3]:

1. Input layer of size 32*32
2. Convolutional layer with 6 kernels of size 5*5 with a stride of 1
3. Average pooling with size 2*2 and a stride of 2
4. Convolutional layer with 16 kernels of size 5*5 with a stride of 1
5. Average pooling with size 2*2 and a stride of 2
6. Fully connected convolutional layer of 120 units of size 1*1
7. Fully connected layer of 84 units
8. Fully connected softmax output layer of size 10

One difference between our CNN and the original LeNet-5 is that we use an output layer of size 2, since we only have two classes of “normal” and “attack”. Another difference is that we use ReLU activation functions instead of tanh activation functions used in the original LeNet-5.

3. EXPERIMENTAL RESULTS AND EVALUATION

We followed the same procedures and methods as laid out in the paper [1]. This section gives more details on the followed procedures, the resulting experimental results, and their evaluation.

3.1 Dataset

The data set used in this paper is the KDD99 data set. It contains 41 features extracted from raw TCP dump data of a local-area network (LAN) and labels the data with the following categories [2]:

- Normal: no attack
- DOS: denial of service, e.g. syn flood
- R2L: unauthorized access from a remote machine, e.g. guessing password
- U2R: unauthorized access to local superuser (root) privileges, e.g., various “buffer overflow” attacks;
- Probing: surveillance and other probing, e.g., port scanning.

3.2 Evaluation Metrics

We used the same metrics used in the paper [1] to evaluate our results:

- Accuracy (ACC): Ratio of correct predictions to total samples. We want to maximize this

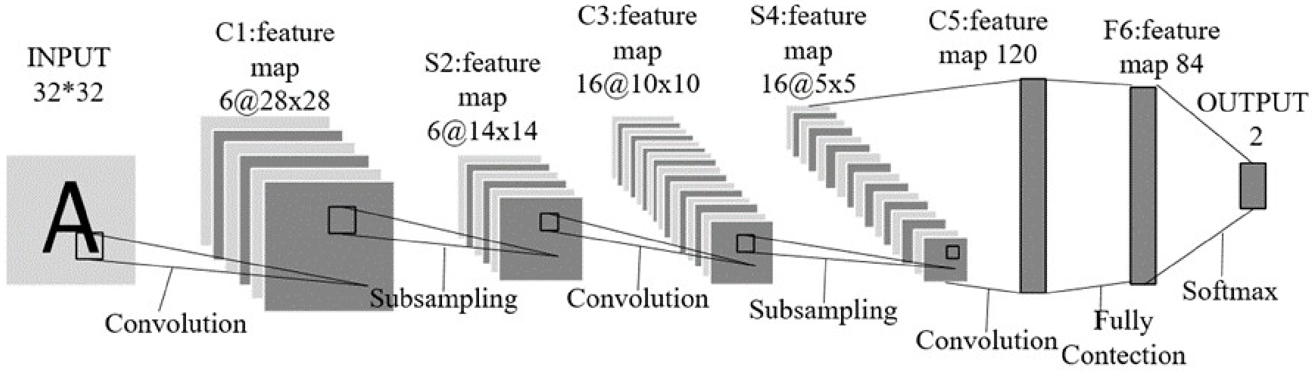


Figure 2. LeNet-5 CNN

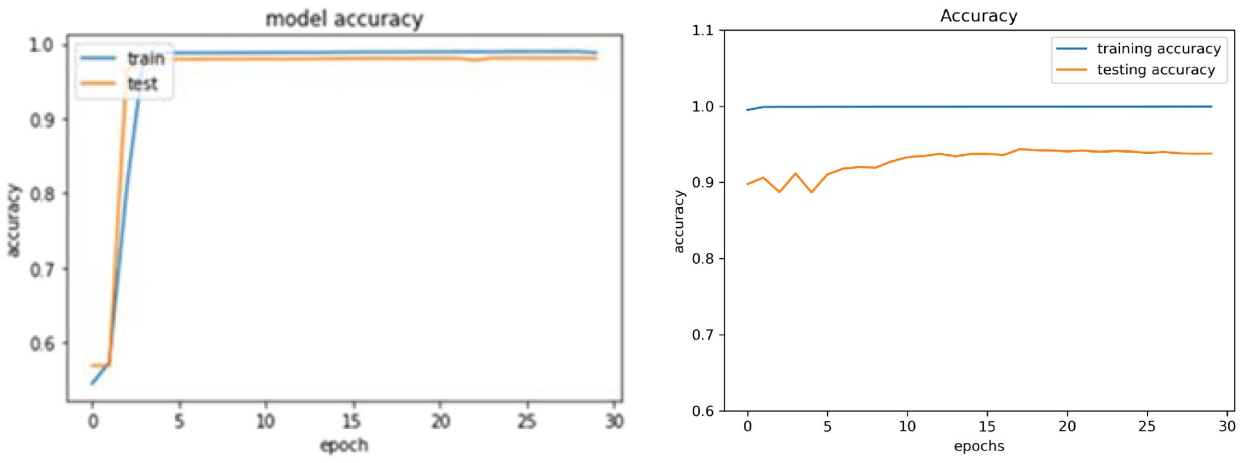


Figure 3. Accuracy per epochs (Left image is from [1])

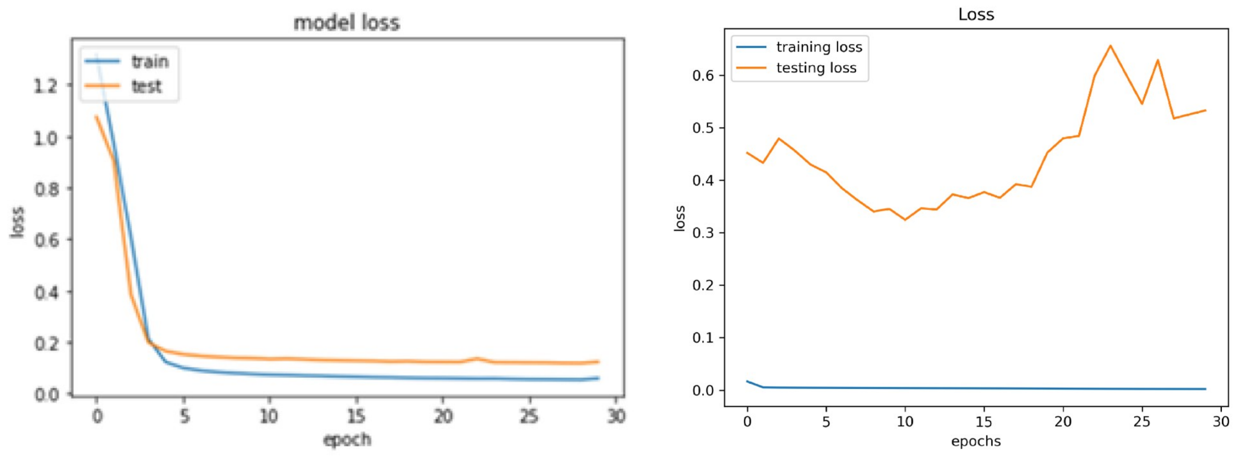


Figure 4. Loss per epochs (Left image is from [1])

- Detection rate (DR): Ratio of correct positive predictions to total positive samples. It gives an indication of how good the model is in detecting attacks. We want to maximize this

- False alarm rate (FAR): Ratio of incorrect positive predictions to total negative samples. It gives an indication of how often the model mistakes a normal sample with an attack. We want to minimize this

3.3 Evaluation Metrics

We ran the experiments on a Windows 10 Laptop with Intel Core i7-4720HQ CPU @ 2.6 GHz, 16 GB RAM, and Nvidia GeForce GTX 970M Graphic Card. Python 3.7 with Tensorflow 2.1 was used to code the system.

The paper [1] performed two sets of experiments:

1. To see the effect of increasing the epochs on accuracy and loss for training and test datasets
2. To see the effect of changing learning rate on all metrics for training and test datasets

We ran the same sets of experiments to test the validity of the reported results in paper [1]. Figure 3 and 4 shows the paper [1] results compared to our results for the first set of experiments. Paper [1] results are on the left side, and our results are on the right side.

Training accuracy and loss are similar except for their initial values. Paper [1] training accuracy starts around 0.6 and quickly rises to 0.99 while our training accuracy starts from 0.99 and stays the same. Paper [1] training loss starts at around 1.2 and falls down to around 0.1 by the epoch 5, while our training loss stays at the same low value of about 0.002. Testing accuracy and loss are very different. Paper [1] testing accuracy and loss follows the training accuracy and loss closely within some narrow margin, while our testing accuracy and loss are largely separated from the corresponding training curves with much wider margins.

To find a better value for epoch paper [1] tested the metrics every 10 epoch for 80 epochs. Their results are in Table 1 and our results in Table 2.

As was evident in the accuracy and loss graphs, our results are lower than the paper [1]. Our highest ACC was 0.946644 at epoch 40, while it was 0.99783 at epoch 80 for paper [1]. Our highest ACC was 0.946644 at epoch 40, while it was 0.99783 at epoch 80 for paper [1]. Our highest DR was 0.951942 at epoch 50, while it was 0.9974 at epoch 80 for paper [1]. Our lowest FAR was 0.023600 at epoch 20, while it was 0.00038 at epoch 80 for paper [1]. Generally we didn't see the improving effect of increasing epochs that was reported in [1].

For the second set of experiments that meant to determine the best learning rate, we selected 5 representative learning rates (we made sure that their best reported learning rate 0.1 was included) for testing, to make the comparisons easier to understand. Paper [1] results are shown in Table 3, and our results in Table 4.

Comparing Tables 3 and 4 we see that our highest ACC was 0.94585 at rate 0.1, while it was 0.99958 at rate 0.35 for paper [1]. Our highest DR was 0.93700 at rate 0.225, while it was 0.99953 at rate 0.35 for paper [1]. Our lowest FAR was 0.01401 at rate 0.005, while it was 0.0002 at rate 0.1 for paper [1]. The paper [1] recommended learning rate 0.1, stating that it resulted in better ACC and DR, but as we saw, it was only superior in terms of FAR, and even its FAR was closely matched by the rate 0.35. So we think that according to their own results they should have recommended learning rate 0.35. Our result is different in that it shows that we can get a good balance of ACC, DR, and FAR using a learning rate of 0.005 (It's ACC and DR are close to best values and has the best FAR), although the values are worse than paper [1].

There seems to be three possible explanations of why we got different results while following the procedures described in the paper [1] closely:

1. The author didn't specify the method used to convert 1D to 2D in the data preprocessing step. Since there are different ways of doing that, it might have led to different results.
2. The author diverged from the LeNet-5 model and made some parameter changes but failed to report it in his paper.

Epoch	**ACC**	**DR**	**FAR**
10	0.99094	0.98959	0.00353
20	0.9948	0.99402	0.00202
30	0.99568	0.99487	0.00099
40	0.99684	0.99624	0.00067
50	0.99743	0.9969	0.0004
60	0.99749	0.99698	0.00041
70	0.9978	0.99739	0.0005
80	**0.99783**	**0.9974**	**0.00038**

Table 1. Paper [1] results of testing every 10 epochs

Epoch	**ACC**	**DR**	**FAR**
10	0.927467	0.905979	0.051046
20	0.941734	0.907068	**0.023600**
30	0.938062	0.915848	0.039724
40	**0.946644**	0.949615	0.056327
50	0.942328	**0.951942**	0.067285
60	0.940290	0.928259	0.047679
70	0.944532	0.927929	0.038866
80	0.941124	0.914413	0.032165

Table 2. Our results of testing every 10 epochs

Rate	**Train ACC**	**Test ACC**	**Train Loss**	**Test Loss**	**Time**	**ACC**	**DR**	**FAR**
0.0001	0.99371	0.98659	0.02425	0.05114	10328.7	0.98659	0.98459	0.00524
0.005	0.99891	0.99268	0.00383	0.01625	7510.3	0.99268	0.99129	0.00167
0.1	0.99982	0.99914	0.00074	0.00391	7543.3	0.99914	0.99898	**0.0002**
0.225	0.99983	0.9994	0.00065	0.00251	7706.6	0.9994	0.99937	0.00045
0.35	0.99988	0.99958	0.00047	0.00175	7524.6	**0.99958**	**0.99953**	0.00022

Table 3. Paper [1] results of testing different learning rates

Rate	**Train ACC**	**Test ACC**	**Train Loss**	**Test Loss**	**Time**	**ACC**	**DR**	**FAR**
0.0001	0.98689	0.83844	0.06230	0.41806	5354.7	0.83844	0.73757	0.07321
0.005	0.99900	0.94317	0.00573	0.32803	5306.8	0.94317	0.90002	**0.01401**
0.1	0.99938	0.94585	0.00266	0.29988	5019.1	**0.94585**	0.90487	0.02282
0.225	0.99939	0.94384	0.00209	0.29606	5108.7	0.94384	**0.93700**	0.06844
0.35	0.99945	0.94452	0.00184	0.31651	5110.7	0.94452	0.92045	0.06240

Table 4. Our results of testing different learning rates

3. Either us or the author had some hidden issues in our setup but failed to notice it.

In either case, our work casts some doubt on the results reported in [1]. These issues can be resolved by obtaining their source code and running it.

4. CONCLUSIONS

Intrusion Detection Systems can greatly benefit from recent advances in Deep Learning techniques such as Convolutional Neural Networks (CNN). Since reproducibility is one of the important metrics in evaluating a scientific study, we reproduced the results of the paper "An Intrusion Detection System Based on Convolutional Neural Network" by Pengju Liu, that uses CNN to detect network attacks from features extracted from network packets. We followed the procedures as laid out in the paper, however we got different results. We conclude that these discrepancies are either due to using different 1D to 2D conversion methods or unreported parameter changes in Liu's paper or some hidden or unaccounted failure on our part or Liu's part. In either case, it calls for a reevaluation of the reported results.

For future work, we suggest obtaining the original source code used in Liu's work and running it.

5. REFERENCES

[1] Liu, P. (2019, February). An Intrusion detection system based on convolutional neural network. In Proceedings of the 2019 11th International Conference on Computer and Automation Engineering (pp. 62-67).

[2] Intrusion Detection Learning. http://kdd.ics.uci.edu/databases/kddcup99/task.html. Web page viewed on March 27, 2020.

[3] Rizwan, M. (2018). LeNet-5 – A Classic CNN Architecture. https://engmrk.com/lenet-5-a-classic-cnn-architecture/. Web page viewed on April 3, 2020.