Deep Learning for Network Flow Analysis and Malware Classification

R.K. Rahul[1,2(✉)], T. Anjali[1,2], Vijay Krishna Menon[1,2], and K.P. Soman[1,2]

[1] Centre for Computational Engineering and Networking (CEN), Amrita School of Engineering, Coimbatore, India
iamrkrahul@gmail.com, anjukrishnadas@gmail.com
[2] Amrita Vishwa Vidyapeetham, Amrita University, Coimbatore, India
m_vijaykrishna@cb.amrita.edu, kp_soman@amrita.edu
https://www.amrita.edu/center/computational-engineering-and-networking

Abstract. In this paper, we present the results obtained by applying deep learning techniques to classification of network protocols and applications using flow features and data signatures. We also present a similar classification of malware using their binary files. We use our own dataset for traffic identification and Microsoft Kaggle dataset for malware classification tasks. The current techniques used in network traffic analysis and malware detection is time consuming and beatable as the precise signatures are known. Deep learned features in both cases are not hand crafted and are learned form data signatures. It cannot be understood by the attacker or the malware in order to fake or hide it and hence cannot be bypassed easily.

Keywords: Network application identification · Protocol classification · Malware classification · Deep learning · Convolutional Neural Network · CNN · Auto encoder

1 Introduction

The scale and density of network traffic is rapidly growing through the years. The protocols which are designed grossly based on TCP-IP model established in the initial days of Internet, lack the necessary features required for such traffic analysis. Most of the protocol classification systems today mainly depends on the parameters such as Port numbers, static headers, IP addresses etc. But, as new protocols, which are being designed every day, are not following the rule of port registration, the situation is worsing for traffic analysers and network administrator [12].

When we take the case of network applications, the traditional way to classify them using meta traffic information was based on limited behavioral properties which are used to define heuristics features. These features again include port numbers, transmission rate and frequency, application and protocol header information etc. [18]. With the advent of mobile and web applications, this scenarios

S.M. Thampi et al. (Eds.): SSCC 2017, CCIS 746, pp. 226–235, 2017.
https://doi.org/10.1007/978-981-10-6898-0_19

is at its worst. Along with this, administrators also face issues like tunneling, random port usage, proxy and encryptions that makes detection and classification almost impossible [16].

Similarly traditional classification of malware is done mainly with heuristic and bahavioral signatures that grapple to keep up with malware evolution. A malware signature is an algorithm or hash that uniquely identifies a specific virus. It is proved that all viruses in a family share common behaviour and a single generic signature can be created for them. However, malware authors always try to confuse antivirus software by writing polymorphic and metamorphic malware that constantly change known signatures and thus fool the system. To avoid all such contempt of behaviors, a flow and code feature based analysis or *data driven* analysis is mandatory for network applications, protocols and malware. Behavioral signature can be mocked, copied, changed or tampered with, but data signatures are abstract and cannot be manipulated that easily [4].

In 2015, Microsoft hosted a competition in Kaggle with the goal of classifying malware into their respective families based on the their content and characteristics. Microsoft provided a set of malware samples representing 9 different malware families. Each malware sample had an ID, a 20 character hash value is used to uniquely identify the sample and a class, an integer label representing one of the 9 malware family (class) to which the malware belong: (1) Ramnit, (2) Lollipop, (3) Kelihos_ver3, (4) Vundo, (5) Simda, (6) Tracur, (7) Kelihos_ver1, (8) Obfuscater. ACY, (9) Gatak [7]. The dataset includes files containing hexadecimal representation of malwares' executable machine code. Each files is composed of Byte Count, Address, Record type, Data and Checksum.

Together all three, can be defined as a multi class classification problem, to make it machine learnable. Selection and processing of the right features from a frenzy of unintelligible data is a near impossible task which makes the above problems an ideal case for applying deep learning [4].

Deep learning is a new subversive machine learning strategy, where extraction of features is done by the machine itself from the given data for the best classification possible. These feature are at best, non orthogonal and significantly enhance the accuracy of classification or regression, compared to human hand crafted features [8,14]. Some supervised learning algorithms include logistic regression, multilayer perceptron, deep convolutional network etc. Semi or unsupervised learning include stacked auto encoders, restricted Boltzmann machines (RBMs), deep belief networks (DBNs) etc. [13,15].We approach the above problems with a convolutional neural network (CNN) with auto encoders and tweak the network performance.

A Convolutional Neural Network (CNN) is a form of feed-forward neural network in which the connection between its neurons is similar to the structure of the animal visual cortex, whose individual neurons are organized in such a way that they respond to overlapping regions tilling the visual field [1,6]. CNNs are composed by three types of layers such as fully-connected, convolutional and pooling. CNN has the ability to see any data as an image and this characteristic allows users to encode certain properties into the architecture. CNN will convolve several small filters on the input image and subsample this space of filter

activations and repeat these processes until we left with enough high level features. Then it will apply a standard feed-forward neural network to the resulting features [2].

The other main method used for feature extraction are auto encoders. They are made to generate a set of features which can be reverse transformed to yield back the original input. This is called bidirectional training. The networks has the same input and output from which it back propagates and learns [6]. In an essence these can be used for kernel type feature mapping normally used with non-linear or non-separable data in traditional machine learning

2 Methodology and Reasoning

A lot of literature is available on signature based network application and protocol classification and also based on statistical features and machine learning. They are all some form of hand crafted features which are time consuming, beatable and inflexible. These methods fail to detect or classify an unknown application and protocol due to the same reason. Zhanyi Wang introduced deep learning in traffic identification [17] which motivated us to take up this work. He has classified applications and protocols using features which are automatically extracted using an auto encoder. The full payload from the network data packet is given to an auto encoder for feature extraction and classification is done by a fully connected dense layer at the end.

The malware classification also have the above mentioned flaws in using the manually handcrafted features for classification. Deep learning has been used for classification of Kaggle data. The winners of Kaggle Microsoft malware challenge have extracted mainly three important features from it like Opcode 2,3 and 4-grams, Segment line count and Asm file pixel intensity features. They are getting an accuracy of 99.98% which might fail while classifying polymorphic and metamorphic malwares and families. Sequence classification methods which is related to gene classification in computational biology [5] have also been proposed. But it too relies on features which are handcrafted. Besides, if old malware is rebuilt to create new malware binaries then their code would be very much alike [9].

2.1 Protocol Classification Using Metadata

In order to collect packet data, Wireshark and Tshark is used. Wireshark [10] is an open source software for analyzing network packet. Only HTTP, SSL, and SMTP protocol packets are selected from the entire collection of captured packets. Since the classification process with entire payload is not computationally easy, only the metadata or packet attributes are taken for the experiment. The metadata contain a partial information about the payload. The collected data packets are converted into comma separated values and it acts as the input to the deep learning architecture. The data packetstrimmed to a uniform length vector of 1024 bytes and the data is converted to decimal format, so it can be easy fed to a network programatically [12].

2.2 Payload Data Collection and Data Preprocessing for Network Application Classification

Since considerable results were obtained for classification using metadata, we extended the experiment to a higher level. Classification of network applications using full payload is done to obtain better results as more data is involved here. We collected the complete payload using `tcpdump` [3] and extracted it. When called, `tcpdump` actually prints headers of each packet and the data of each packet including its link level header. From that, only the payload information of three different network applications were collected. Browsers, Facebook and Torrent are the three classes of applications chosen. The browser class consists payloads of both Opera and Mozilla Firefox. Among the total of 33,268 packets, the first class contains 17,024 packets of browser payloads, second class contain 8528 Facebook payloads and in the third class 7,716 packets of torrent application payloads. The data payloads are originally in the form of hexadecimal values and it is further converted into decimal values for the purpose of feeding it into deep learning network as mentioned before.

2.3 Malware Classification Using Kaggle Data

The malware data provided by the Microsoft in Kaggle contain 9 families. The main objective is to classify these to their respective families. Each observation is a representation of the file's binary content. This hexadecimal data is preprocessed to decimal in order to feed it to the CNN. The preprocessed data is a vector of 128 values, each coma separated. The preprocessed data is fed into a Convolutional neural network with two convolutional layers along with max pooling layers followed by two dense layers. 64 one dimensional filters are used in first convolutional layer and 32 two dimensional filters are used in second convolutional layer. Architecture is as given in the Table 1.

2.4 Different Convolutional Neural Networks Implemented

A four-layer convolutional neural network was implemented with two convolutional and two fully connected layers. The network architecture used for the classification of protocols, network applications, and malware is given in the Table 1 respectively with weight dimensions. The input data sample size fed into the network, size of filters of first two convolutional layers, and that of fully connected layer and output layer are also given by the Table 1.

The neural network is expected to learn these filter weights over the training process such that it extracts the essential features from the data samples which are able to distinguish the different classes. The rectified linear activation function (RELU) was chosen as the activation function for both convolutional layers. Then the information being extracted from these features are used to predict the label corresponding to each data point by adjusting weights and biases across the two fully connected layers [11]. In order to avoid over fitting, dropout was enabled. We have chosen the Googles TensorFlow® framework to implement our network.

Table 1. CNN architecture used in the Classification Processes for Specific Tasks

	Size of input vector	No of filters in 1st convolutional layer	No of filters in 2nd convolutional layer	Neurons in fully connected layer	Output layer
Protocol	1024	128	64	8	3
Application	2048	128	64	8	3
Malware	128	64	32	16	9

2.5 Implemented Autoencoder Architecture

The entire payloads were fed into an auto encoder and features were extracted. The architecture of the auto encoder used in the experiment is shown in the Fig. 1. The packet attributes of three protocols is given to the designed auto encoder.The network has an input of length 1024 and a 512 node middle layer. *tanh*, is used as the activation function in all the three layers. The loss function for training was taken as the root mean square error between the outputs of final nodes and the inputs. The network is trained using batched stochastic gradient decent, which is faster than individually updating after each data. The middle layer samples were taken as input and are fed into the CNN which is further trained with data labels as the ground truth.

Feature selection from the auto encoder is computationally heavy in the training stage but is a one time process. Once the network has been trained,

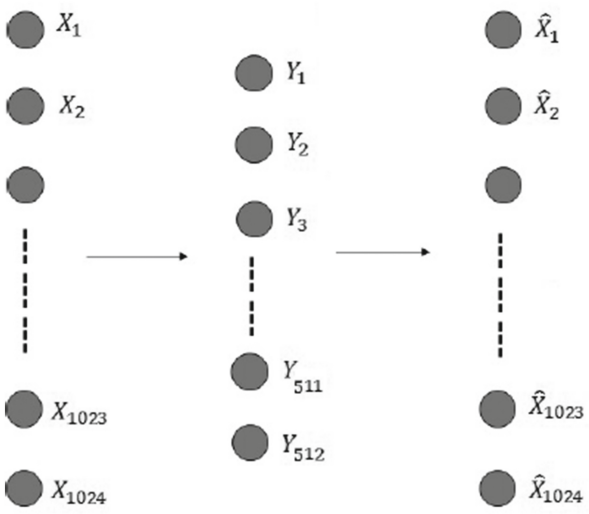

Fig. 1. Architecture of Autoencoder used for protocol classification

features can be extracted from data frames with simple computations such as matrix multiplication.

The only difference between the architectures of auto encoders used for application classification and protocol classification is the input vector size; 2048 for the application classifier. The number of nodes in the hidden layer is 512 as in the above case. The samples from middle layer was taken as the features for classification, to be fed to into the CNN.

3 Results and Discussions

3.1 Protocol Classification

The preprocessed samples of metadata for three different protocols are given to the CNN. Out of 75,000 data given 52,500 packets of data are given for training and remaining 22,500 are given for testing. The result shows classification on test data. 83.78% accuracy is obtained for 2000 iterations. The same data set is given to the auto encoder with a softmax layer for classification. However, this classification gave less accuracy with a best measure of 75.57 for 1700 iterations.

3.2 Network Application Classification

The data contains payloads from three different network applications are given to the CNN mentioned in the Table 1. Different parameters of the CNN were changed and the changes in the accuracy were observed.

In the experiment stage, we tried with three different learning rates. Initially it was fixed at 0.01 and an accuracy was 20.12% for 1000 epochs. Then we changed the learning rate to 0.001 and obtained accuracy of 45.20% for the same number of iterations. So we again decreased the learning rate to 0.0001 and got a high accuracy of 84.26%. Then we fixed the value of learning rate as 0.0001 and then changed the dropout values.Since the training takes much time here the number of epochs is fixed to 200. The value of dropout was changed from 0.1 to 0.9 gradually and we could observe an evident increase in accuracy. The accuracy for 0.1 dropout is 62.50% and for 0.9 it is 91.90%. We fixed the value of dropout as 0.9 for our architecture. After choosing the values for learning rate and dropout, we changed the number of epochs from 200 to 2000 and observed the results. The corresponding observations are plotted in Fig. 2. Here, we can observe that for 2000 epochs the accuracy obtained was 95.50%. The class-wise accuracy for 3 different epochs are given in the Fig. 3. All these results were obtained by the classification using only CNN . The next type of classification combines an auto encoder with the existing CNN. The data points were directly fed into an auto encoder and the features were piped to the CNN. The result obtained for these two methods were compared to the existing results [17] and are given in the Table 2.

The accuracy obtained for feature extracted CNN is high compared to the other two methods, giving a class wise accuracy for both browsers and chat

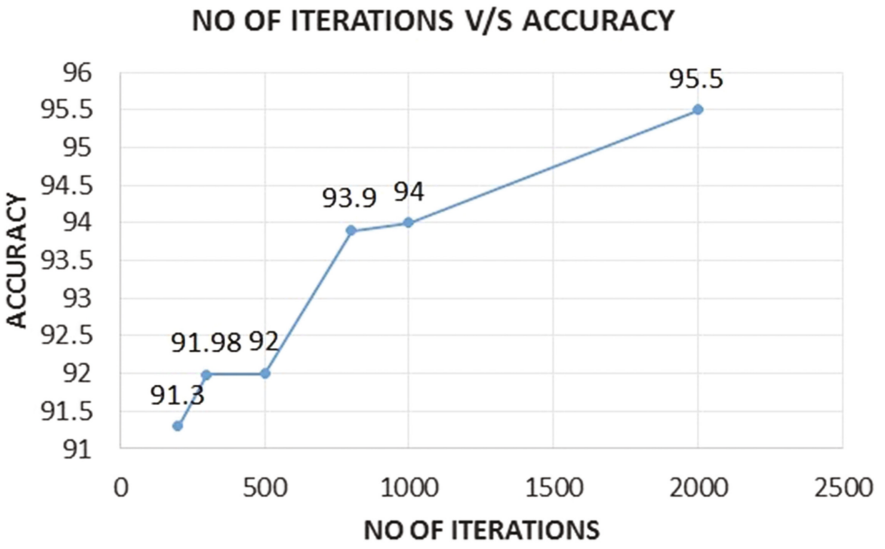

Fig. 2. Training epochs vs Accuracy graph with a learning rate of 0.0001 and dropout of 0.9

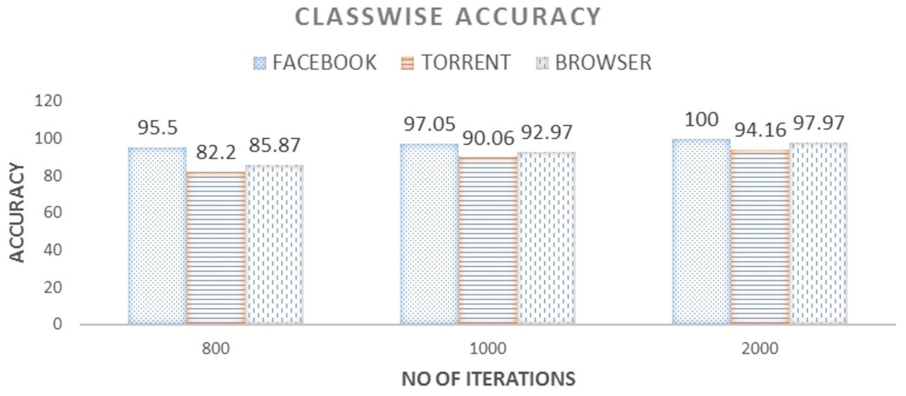

Fig. 3. Class-wise accuracy for three different epochs

Table 2. Results of application classification

Class	Existing results(%) [17]	Accuracy for CNN(%)	CNN + Auto encoder(%)
Browser	88.60	97.97	100
Chat	99.80	100	100
Torrent	98.70	94.16	98.90
Overall	95.70	97.37	**99.63**

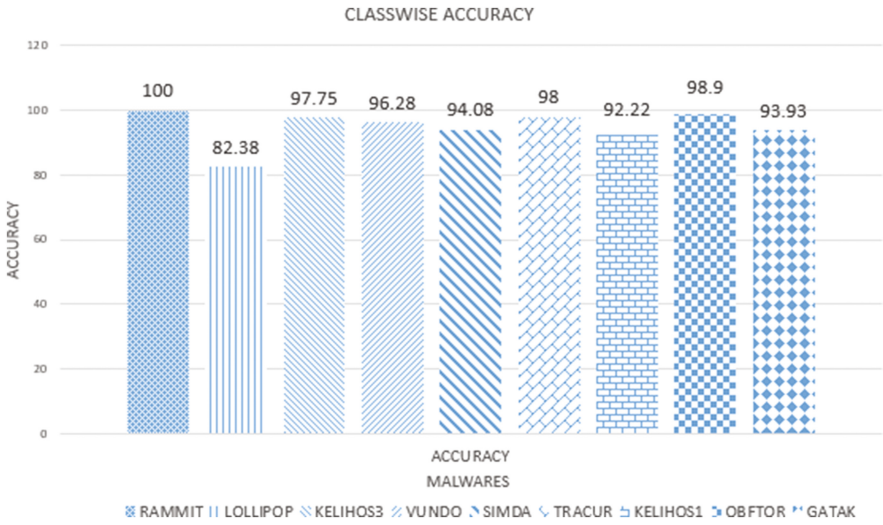

Fig. 4. Class-wise accuracy for malware classification

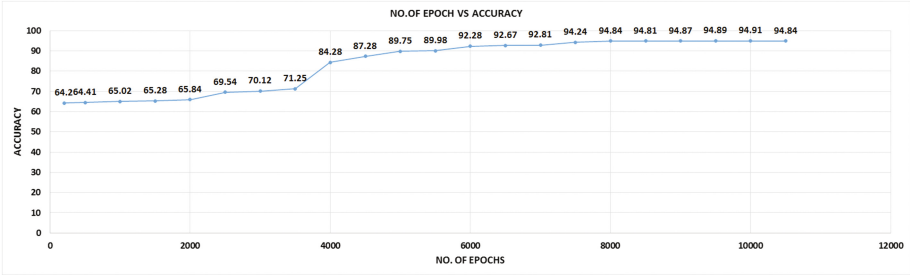

Fig. 5. Training epochs vs Accuracy graph for malware classification with learning rate of 0.001 and dropout of 0.9. Average Testing Accuracy is 94.91%

applications at 100% and that of torrent(with encryption enabled) is 98.9%. The overall accuracy is at 99.63% where as that of the existing method [17] was at 95.70% and classification using CNN only is at 97.37%.

3.3 Malware Classification

Totally 9 families of malware is taken for classification. The CNN architecture with two convolutional and two dense layers give maximum accuracy for the data sample (full Kaggle dataset was too big to process, so random sample was taken). The average accuracy of 94.91% is achieved with this architecture as shown in Fig. 5. The class wise accuracy is also shown in the Fig. 4.

4 Conclusion

Deep learned features are abstract in nature and cannot be attributed to any specific measure of the data entities (like traditional features which are hand crafted), such as network traffic and malware that generate huge amounts of data. From the results of our experiment we can conclude that this data in each case has got more information than what is humanly visible, like basic statistical behaviours, port associations, header information and format etc. The obvious benefits are that these features are also invisible to the attacker and data fingerprints like these cannot be manipulated. For malware we have used their binary executable code and for the network traffic we used the transmitted payload inside each packet. We speculate that since the code profiles seldom change for even the most tricky polymorph, a static pre-trained model will do that can be integrated with the firmware. The case with network traffic is almost very similar. Newer and proprietary protocols/applications are mostly derived from existing ones and can be caught since their flow signatures will remain more or less static. We also observed that among all the three, torrent gives the least accuracy for which we speculate that it is due to the tunnelling and encrypting behaviour of torrent transmissions. But given these conditions, we still believe we can identify them accurately in real time, if we pump in more data to train the network.

5 Future Work

Presently the classification was done only using CNN and auto encoder. In future, it can be further extended to RNN and LSTM as transmitted data might have some auto correlations or sequential behaviour. The experiment was done by collecting packets over a small network which can be expanded over larger ones. We can try for more applications especially the ones with proprietary protocols. Classification can be performed on Botnets to identify infections in real time. The malware classification has a lot of room for improvement, and can chunk more data toward this goal. The main use of this malware model is with in disassemblers or firmware profilers which can see the actual code passed for execution. Any code suspected to be malicious can be filtered or at least be quarantined prior to the real execution of it. In the same way a network traffic filter can be set up on bridges and routers based on learned models trained on malicious or congestion causing traffic to do selective load shedding.

References

1. Convolutional neural network. https://en.wikipedia.org/wiki/Convolutional_neural_network. Accessed 10 May 2017
2. Deep learning. https://en.wikipedia.org/wiki/Deep_learning. Accessed 29 Nov 2016
3. Tcpdump. http://www.tcpdump.org/tcpdump_man.html. Accessed 27 Apr 2017

4. Anjali, T., Menon, V.K., Soman, K.P.: Network application identification using deep learning. In: 6th IEEE International Conference on Communication and Signal Processing (2017, accepted)
5. Drew, J., Moore, T., Hahsler, M.: Polymorphic malware detection using sequence classification methods, pp. 81–87 (2016)
6. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
7. Microsoft: Kaggle malware data. https://www.kaggle.com/c/malware-classification/data. Accessed 11 May 2017
8. Nagananthini, C., Yogameena, B.: Crowd disaster avoidance system (CDAS) by deep learning using extended center symmetric local binary pattern (XCS-LBP) texture features. In: Raman, B., Kumar, S., Roy, P.P., Sen, D. (eds.) Proceedings of International Conference on Computer Vision and Image Processing. AISC, vol. 459, pp. 487–498. Springer, Singapore (2017). doi:10.1007/978-981-10-2104-6_44
9. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, VizSec 2011, pp. 4:1–4:7. ACM (2011). http://doi.acm.org/10.1145/2016904.2016908
10. Orebaugh, A., Ramirez, G., Beale, J.: Wireshark & ethereal network protocol analyzer toolkit (2006)
11. Athira, S., Mohan, R., Poornachandran, P., Soman, K.P.: Automatic modulation classification using convolutional neural network. IJCTA **9**(16), 7733–7742 (2016)
12. Rahul, R.K., Menon, V.K., Soman, K.P.: Network protocol classification using deep learning. In: 6th IEEE International Conference on Communication and Signal Processing (2017, accepted)
13. Soman, K., Diwakar, S., Ajay, V.: Data Mining: Theory and Practice [WITH CD]. PHI Learning Pvt. Ltd., Delhi (2006)
14. Soman, K., Loganathan, R., Ajay, V.: Machine learning with SVM and other kernel methods. PHI Learning Pvt. Ltd., Delhi (2009)
15. Team, T.D.: Deep learning tutorials. http://deeplearning.net/tutorial/. Accessed 29 Nov 2016
16. Tongaonkar, A., Keralapura, R., Nucci, A.: Challenges in network application identification (2012)
17. Wang, Z.: The applications of deep learning on traffic identification. BlackHat USA (2015)
18. Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning, pp. 250–257 (2005)