IEEE

JCT College of Engineering and Technology

Approved by AICTE,New Delhi & Affiliated to Anna University Chennai.
Pitchanur,Coimbatore-641105.

TÜV AUSTRIA
CERTIFIED
EN ISO 9001
TÜV AUSTRIA CERT GMBH

CERTIFICATE OF PRESENTATION

This is to certify that

Mr. HEMANTH TEJA

has presented a paper titled

EMBER - ANALYSIS OF MALWARE DATASET USING CONVOLUTIONAL NEURAL NETWORKS

at the 3rd International Conference on Inventive Systems and Control (ICISC 2019) organized by JCT College of Engineering and Technology during 10-11, January 2019 at Pichanur, Coimbatore, Tamil Nadu, India.

IEEE ISBN: 978-1-5386-3950-4

Session Chair

P. Pitchandi

Conference Chair
Dr. P. Pitchandi

Principal
Dr. G. Ramesh

JCT INSTITUTIONS
ENGINEERING & POLYTECHNIC
APPROVED BY AICTE NEW DELHI, AFFILIATED TO ANNA UNIVERSITY CHENNAI
PICHANUR COIMBATORE TAMILNADU INDIA

3rd International Conference on Inventive Systems and Control [ICISC2019]

Letter of Acceptance

Date : 19 Dec 2018
Presentation Type : Oral Presentation
Accepted Title : Ember - Analysis of malware dataset using convolutional neural networks
Authors Name & Institution Name: **Subhojeet Pramanik, Hemanth Teja**
School of Computing Science and Engineering, VIT University Chennai Campus, India
Paper ID : ICISC 101

Dear Author
It is our pleasure to inform you that our Conference Committee has accepted your refereed research paper for the oral presentation at **3rd International Conference on Inventive Systems and Control [ICISC 2019]** being held on 10-11 January 2019 at JCT College of Engineering and Technology, Coimbatore, Tamil Nadu, India.

The main theme of 3rd International Conference on Inventive Systems and Control [ICISC 2019] is facilitating, fostering and harnessing the recent innovations in the area of smart systems and control techniques to meet the key challenges of the rapidly developing smart society.
The deadline of the full paper submission is 27 December 2018 Papers meeting the IEEE standards are set forth to publish in the conference proceedings and will be recommended for publication in IEEE Xplore Digital Library. All the conference participants are expected to pay the registration fee according to the conference policy.
We are looking forward for your participation in 3rd International Conference on Inventive Systems and Control [ICISC 2019] at JCT College of Engineering and technology to give an oral presentation on your potential research work.

Sincerely,

Prof Dr. P. Pitchandi,
JCT college of Engineering and Technology,
Coimbatore, India.
profpitchandi@gmail.com , pichandi.icisc17@gmail.com

TECHNICAL SPONSOR
IEEE

EMBER – Analysis of Malware Dataset Using Convolutional Neural Networks

Yanambakkam Hemanth Teja
School of Computing Science and Engineering
Vellore Institute of Technology, Chennai
Chennai, India
yhemanthteja@gmail.com

Subhojeet Pramanik
School of Computing Science and Engineering
Vellore Institute of Technology, Chennai
Chennai, India
subhojeet.pramanik2015@vit.ac.in

***Abstract* – The aim of this research is to implement Neural Network algorithms for investigating malevolent Windows portable execution files. The paper utilizes EMBER – a benchmark dataset that contains features extracted from a million binary files. The dataset contains training samples (malicious, benign and unlabeled samples) and test samples, providing numerous cases to build models that enhance information security. We implemented various neural network architectures using convolutional and fully connected layers to classify a given PE file as malware or benign. Our proposed neural network architecture achieves higher performance than the existing gradient boosting based baseline models. The results achieved are assembled and compared based on precision, recall and f1-score.**

Keywords—Portable Execution Files, Binary Files, Information Security, Convolutional Neural Networks, Feed Forward Neural Network

I. INTRODUCTION

The various supervised machine learning models can be used as appealing tools to comprehend complex relations between the various file attributes and differentiate between malicious and benign samples. Given the accessibility of rich benchmark datasets, we aim to construct deep learning models for malware detection and contribute to the open research community. With the assistance of Endgame Malware Benchmark for Research (EMBER) dataset, we have effectively built models for prediction. The dataset contains samples released with the sha256 hash of the original file, and a label that denotes whether the file is malicious or benign. Furthermore, given the issues of security and the tedious procedure of deciding if a given document is infected, even the well-trained models fail to achieve the best of accuracies. Hence, this paper addresses these issues by using connectionist neural networks to attain significant accuracies with respect to f1-score and recall.

The 900K training samples contain 300K malicious, 300K benign and 300K unlabeled samples. Furthermore, the 200K test samples contain 100K malicious and 100K benign samples. The appropriately regularized neural network models evaluate and train neural network architecture on the Ember dataset and generate a similar distribution of the features and labels corresponding to the training data. Using existing source codes of benchmark classifiers and integrating various deep learning models, this paper aims at providing algorithmic models actualized to accomplish and achieve best outcomes.

II. BACKGROUND

We summarize the important context in the execution of portable execution file format in sections A and B. Section A provides an overview to the format of a PE file. Furthermore, section B describes the dataset variables while section C describes the important features of the dataset that have been utilized for modelling.

A. PE File Format

The PE file format [1] depicts the wrapped executable data structures that contain information necessary for Microsoft Windows framework, and incorporates executable dynamically linked libraries (DLLs), and FON files. The main components of the PE file include date/time Stamp, the file pointer, linker, object file, relative virtual addressing, virtual addressing, sections and file headers. Furthermore, the file header includes attributes such as- MS-DOS stub (image only), signature (image only) and COFF file header (object and image). This standard arrangement of headers followed by sections is at present utilized in Intel, AMD and other ARM architectures.

PE segments contain code and instated information that the Windows loader uses. The loader maps executable memory pages, individually. Additionally, it can also map imports and exports that are characterized by the file-records. The sha256 hash of the original file is used as a distinctive identifier.

Referring the 32-bit layout of the PE file from Fig. 1, we can summarize some of the important functionalities as [2]:

- File Pointer – locates units/modules within a file.
- Linker – processes units/modules of object files.

- Sections – segments that contain basic code/data of a PE/COFF file.
- Signature- 4-byte specification validating PE format.

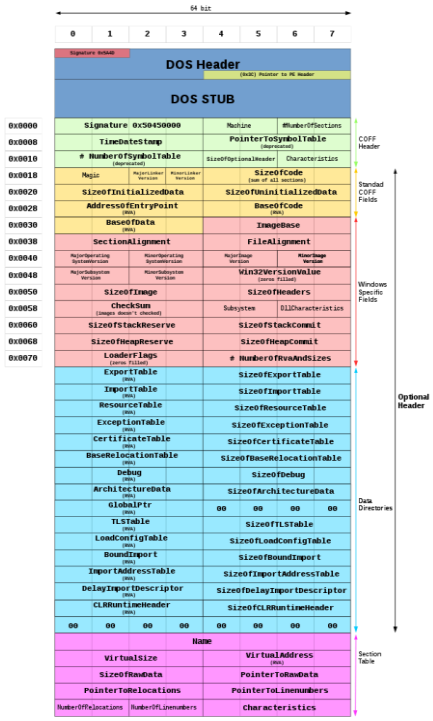

Fig. 1. The layout of 32-bit Windows portable executable file

Optional header pointers include debug information, resources, exceptions and summarized information of the PE executable file [5].

B. Dataset Desciption

The Endgame Malware Benchmark for Research dataset contains four most important attributes that are used for modelling. The dataset is a collection JavaScript Object Notation (JSON data format) objects wherein each line of the dictionary contains an individual, language-independent JSON object.

These independent objects include:

- Sha256 value as a distinctive attribute of a file.
- Coarse time information – provides details about when a given file was opened.
- Groups of raw features.
- Label indication as follows:
  - i. 0 for “benign”
  - ii. 1 for “malicious”
  - iii. -1 for “unlabeled”

The research has utilized an existing code that allows researchers extract numeric features from the raw features. Raw features are used for model-building. The existing model consists of a strategically vectorized matrix for training a base model. A dataset was gathered, and labels were produced utilizing the McAfee virus scanner by Schultz et al. [6]. The training features which contain the unlabeled samples have helped in crafting a semi-supervised neural network. The training and testing features of the dataset have been sorted separately, and the sha256 value is used to interface highlights to raw data.

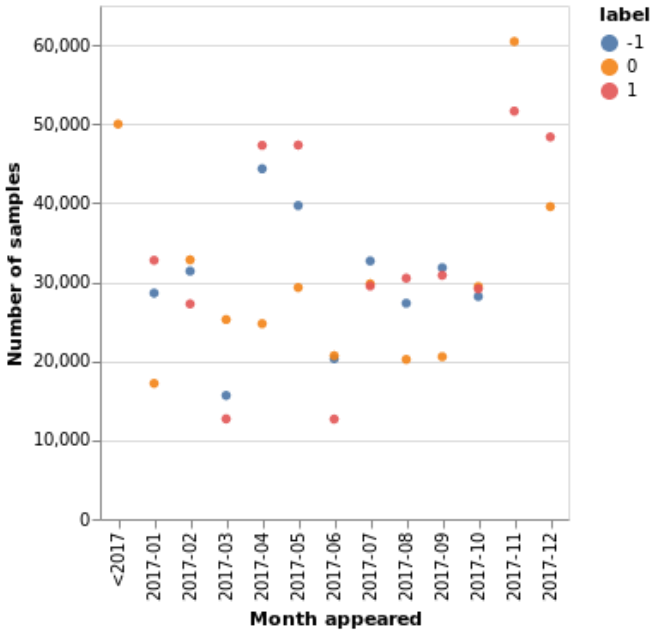

Fig. 2. Graph depicting the quantity of the monthly appearance of the 3 types (benign, malicious, and unlabeled) of samples in 2017

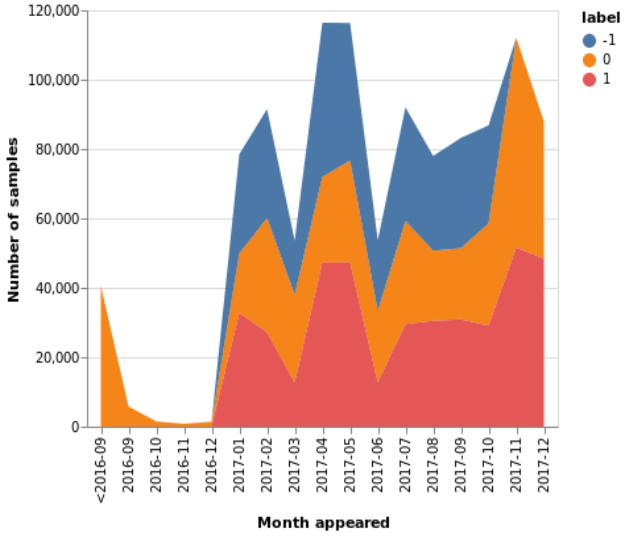

Fig. 3. Graph depicting the quantity of monthly appearance of the 3 types (benign, malicious and unlabeled) of the samples for 2016-2017.

Fig. 2 and Fig. 3 represent the spread of benign, malicious and unlabeled files for the years 2016 and 2017. The visualization was carried out using the Altair library of Python [3].

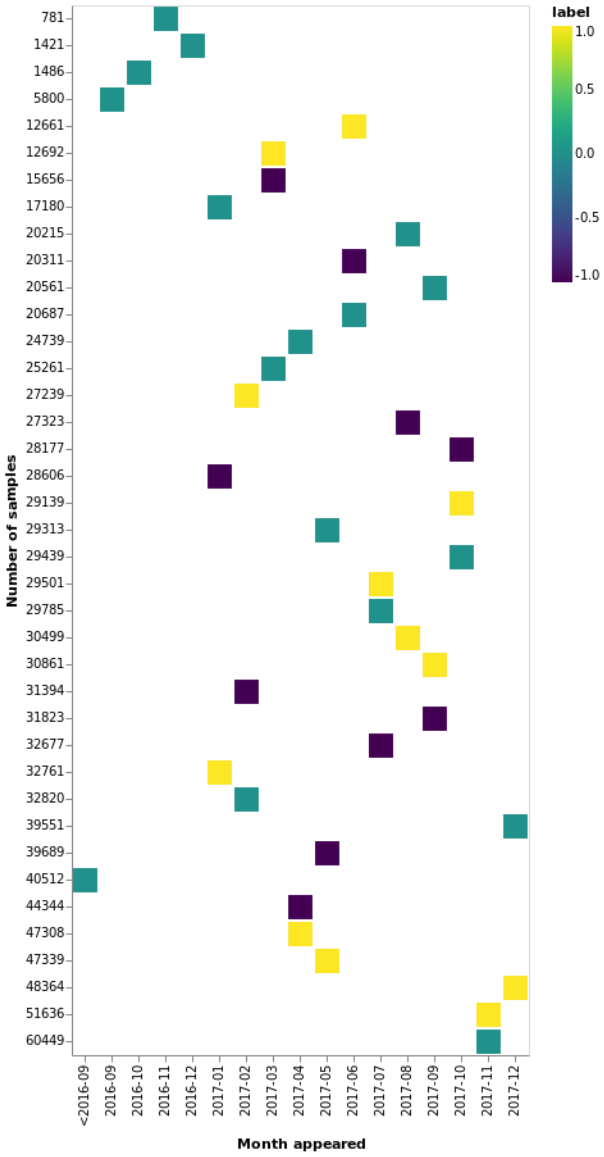

Fig. 4. Heat map of the monthly appearance of the samples in 2016-2017

Fig. 4, depicts a heat-map of the samples for the years 2016 and 2017. From the Heat-map, we can infer that the malicious samples have spread in the year 2017.

A. Dataset Features

The Endgame Malware Benchmark for Research dataset consists of the following important features [3]:

- The common object file format that provides timestamp, list of strings, system and symbol table offset, characteristics and size of optional header.
- Imported functions for standard model drafting.
- Exported functions.
- Sections that provide virtual address, physical address, Pointer to data, Relocations and size.
- DOS header values that provide magic number, pages in the file, address of relocation table, OEM information, PE header offset, overlay number and initial IP value.

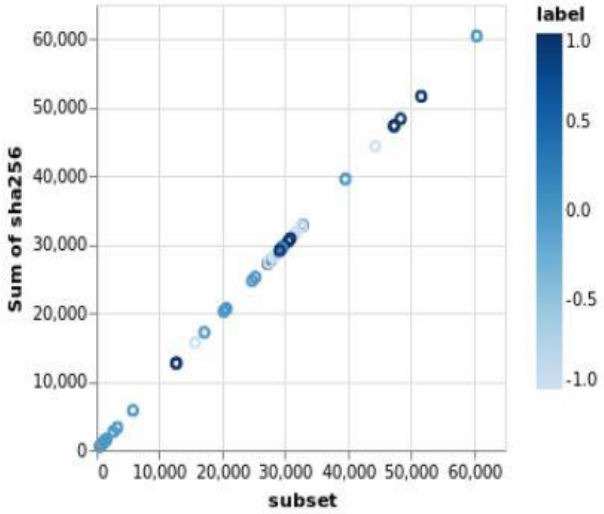

Fig. 5. Monthly appearance Sha256 hash of file and label index in 2016-2017

From Fig. 5, we can infer that the number of malicious samples has reached a peak of about 50000 for the eleventh month of the year 2017.

III. Related Work

Ember provides standard dataset features that can help build standard machine learning models for malware analysis. But standard machine learning models do not seem to provide the best accuracies and many models seem to overlap conceptually [3,7] and one of the most enhanced deep learning classification of malware is discussed in [9]. Deep learning models implemented in this research enhance accuracy through multi-layer processing. The models implemented learn the feature extraction and optimize the features extracted. Although neural systems have been considered for quite a long time, latest advances in power and expanded information volumes have empowered the use of multi-layer neural systems to extensive datasets, which has brought about huge enhancements over conventional machine learning procedures [10]. One of the most important segments of malware analysis involves representing the data more abstractly as the network grows deeper. This can only be achieved through the use of these profound neural systems.

A. Gradient Boosted Decision Tree Results

The baseline model [3] vectorized raw features, which in turn was used to train a gradient-boosted decision tree. While the FP rate was less than 0.1, the model achieved a threshold of 0.871 % and a detection rate exceeding 92.99%. On the contrary, while FP rate was below 1%, the model achieved 98.2 % detection rate. The ROC plot results in 92.2 % detection rate while the false positive rate is less than 0.1 %.

Furthermore, the model achieved a 97.3 % detection rate at less than 1% false positive rate.

IV. PROPOSED MODEL

Code derived static data when combined with conventional methods, fail to detect malware and hence feature extraction and multi-layer processing comes into priority [11]. We propose two different neural network architectures to perform the task of malware classification. Our first architecture is a feed-forward architecture and the second is a convolutional architecture. These architectures involve fundamental machine learning and deep learning. The architectural details of the proposed models are as follows :

A. Feed Forward Classifier

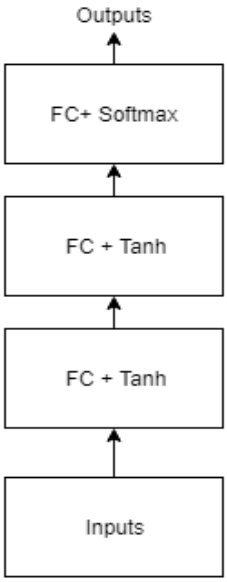

Fig. 6. Feed-Forward Malware classifier

As depicted in Fig.6, the feed forward classifier consists of two fully connected layers of dimension 128 and 64. Each of the layers is used to reduce the input dimensionality of the input feature space from 2351 features. Output of each layer is passed through a tanh function which scales the output activations in the range of -1 and 1. The output of the last hidden layer is passed through a feed forward layer with two hidden units and a softMax activation function. The first two layers also use dropout to improve the regularization performance.

$$f1\_out = \tanh(FC(inputs,128)) \quad (1)$$

$$f2\_out = \tanh(FC(f1\_out,64)) \quad (2)$$

$$Pred = softMax(FC(f1\_output,2)) \quad (3)$$

The overall operations of the feed forward classifier are described by equations (1), (2) and (3).

B. Convolutional Classifier

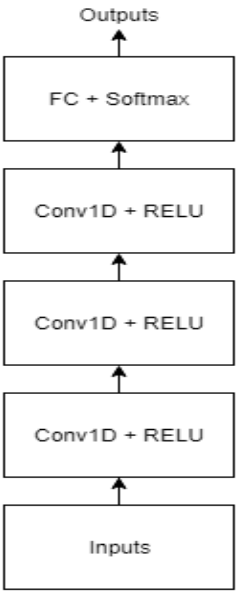

Fig. 8. Convolutional Malware Classifier

Fig. 7 depicts the convolutional classifier that uses layers of convolution followed by a feed forward classifier layer. Since a convolutional layer processes data only for its receptive field, it can drastically reduce the number of parameters used in the computation process. Furthermore, using convolutional layers allows us calculate features from local spatial regions in an input data, which can be of importance for certain input types. The proposed model consists of three layers of convolution. The first layer is 1D-Convolution of kernel size 8, 8 output features and a stride of 2. The kernel size is kept small in the initial layers to reduce the receptive field and capture small variations in adjacent regions of the input data space [12]. The second layer is similar to the first layer, but the number of output channels is increased to 16 and the stride is increased to 4. Higher stride value helps to reduce the computational steps in the convolutional operation.

The third layer is a convolutional layer with kernel size of 15 and output channels of 4. A stride of 5 and a dilation of 10 was used in this layer. The receptive field of the final layer is kept larger than the initial layers to learn robust high-level representations of the input data. Each of the layer output used the RELU activation function to scale the output activations in the range of 0 and 1 and to learn non-linear feature representations. The final layer is a feed forward layer with output dimension ‘2’ and the softMax activation function gives the probability distribution indicating if a PE file is a malware or benign file.

$$Conv1 = Relu(Conv1(inputs)) \quad (4)$$

$$Conv2 = Relu(Conv2(Conv1)) \quad (5)$$

$$Conv3 = Relu(Conv3(Conv2)) \quad (6)$$
$$Pred = softMax(FC(Conv3,2)) \quad (7)$$

The overall operations of the proposed model can be summarized by the equations (4), (5), (6) and (7). We implemented each of the models described above using Pytorch and used an Adam optimizer with a learning rate of 0.01 to train the model. The results demonstrate the performance of the model in the test dataset.

V. STEPS FOR TRAINING

With the end goal to progress malware recognition to a highly scaled model that can react in seconds, we have listed the steps which utilize just short successions of the underlying unique information [13]. The steps are as follows:

- Import necessary python libraries such as – pandas, NumPy pickle, sklearn, tensorBoardX.
- Initialize data directory and the EMBER dataset.
- Initialize global variables such as- batch_Size, train_epochs and learning_rate.
- Load the vectorized features and the metadata.
- Filter malwares that are untagged.
- Construct and initialize train and test data loaders.
- Initialize model training by attaining inputs, zero(ing) the parameter gradients, carrying out forward, backward analysis and optimizing the model.
- Carry out test evaluation.
- Save the training model.

VI. RESULTS AND DISCUSSION

With increasing number of malware files, malware analysts struggle to derive conclusions as they need to extract information from this large-scale data [14]. We have implemented deep learning models and compared the results of feed forward and convolutional neural network classifiers. The implemented models diminish the requirement for feature building, which in turn is one of the characteristics of these profound networks [15]. In order to compare the performance of these two classifiers, the accuracies for both have been measured individually. The precision score along with the f1-score and recall of each model have been attained.

TABLE I. Comparison of Implemented Models

Comparison	F1-score	Precision	Recall
Convolutional Neural Network	0.95	95%	0.95
Feed Forward Network	0.97	97%	0.97

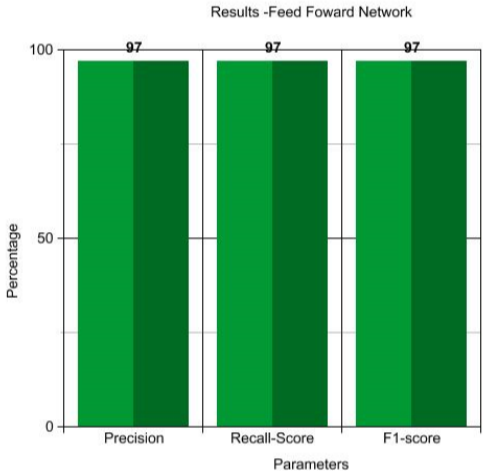

Fig. 8. Result summary – Feed Forward Network

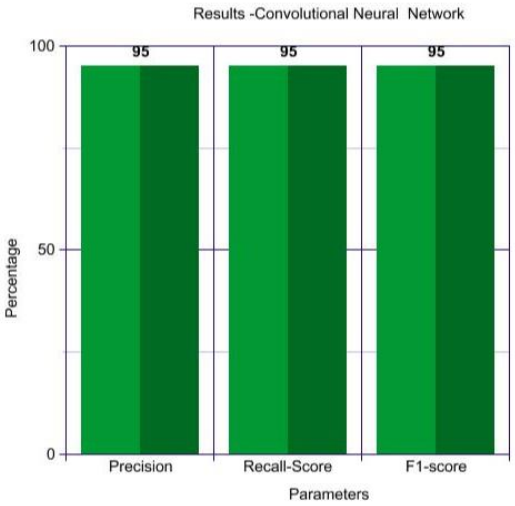

Fig. 9. Result summary – Convolutional Neural Network

From Fig. 8 and Fig. 9, we can conclude that the feed forward network has provided a higher precision with respect to f1-score and recall parameters. As given in Table 1, Feed forward network has achieved a precision of 97 % as contrasted to the convolutional neural network which has achieved a precision of 95%. Furthermore, the average recall and precision scores of the feed forward network are higher than that of the convolutional neural network. The convolutional model has achieved an f1-score of 0.95 and a recall score of 0.95. On the contrary, the feed forward model has achieved an f1-score of 0.97 and a recall score of 0.97. The variation of these scores have been visualized individually for each network.

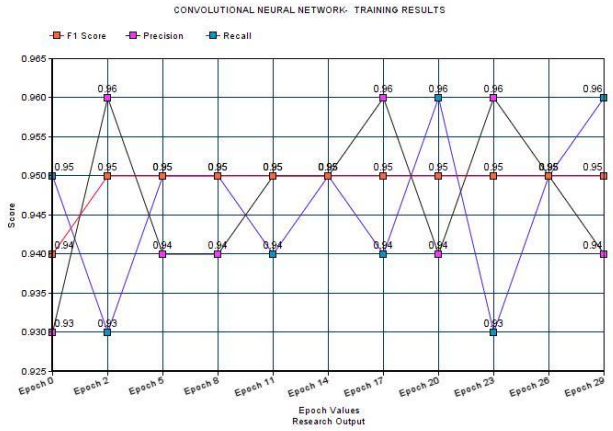

Fig. 10. Comparison of F1-score, precision and recall for convolutional neural network training

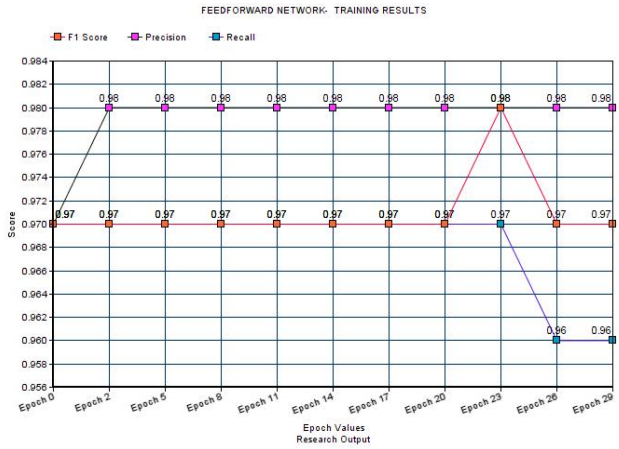

Fig. 11. Comparison of F1-score, precision and recall for feed forward network training

Fig. 10 and Fig. 11 depict the variation of f1-score, precision and recall over various epochs in the convolutional and feed forward model respectively. From the results depicted by the figure, we can conclude that both the models perform better than the baseline model. Both the models result in sufficient amount of recall, which is of utmost importance when developing malware detection systems. The feed-forward model performs better than the convolutional model as there is not much of spectral features in the dataset or the size of the input feature space is low. Convolutional networks work better for datasets that have spectral features and features where local regions are of particular importance. In the case of the ember dataset, the spectral size is one dimensional and has only 2351 features. However, we provide comparison of two very popular deep learning model on the ember dataset. From results it would be safe to conclude that deep learning models are indeed a better alternative to the traditional gradient-boosted method used by previous researchers.

REFERENCES

[1] Wikipedia contributors. (2018, October 19). Portable Executable. In Wikipedia, The Free Encyclopedia. Retrieved 10:50, December 24, 2018

[2] M. Pietrek., “Inside windows-an in-depth look into the win32 portable executable file format,” MSDN magazine, pp. 17(2), February 2002.

[3] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh & Scott Sievert (2018). Altair: Interactive Statistical Visualizations for Python. Journal of Open Source Software

[4] Hyrum S.Anderson and Phil Roth, “Ember: An open dataset for training static PE malware machine learning,” Computer Science Repository, Cornell University Library, pp. 3-6, April 2018.

[5] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq. Pe-miner “Mining structural information to detect malicious executables in real-time,” International Workshop on Recent Advances in Intrusion Detection, Springer, pp. 121–141, 2009.

[6] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, “Data mining methods for detection of new malicious executables,” Security and Privacy, IEEE Symposium, pp. 38–49, 2001.

[7] J. Saxe and K. Berlin, “Deep neural network based malware detection using two dimensional binary program features,” 10th International Conference on Malicious and Unwanted Software, IEEE, pp. 11–20, 2015.

[8] J. Z. Kolter and M. A. Maloof, “Learning to detect malicious executables in the wild,” Machine learning in Python. Journal of Machine Learning Research,” pp. 2825–2830, 2011.

[9] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, “Malware detection by eating a whole,” exe. Arxiv preprint arXiv:1710.09435, 2017.

[10] Quan Le , Oisín Boydell , Brian Mac Namee and Mark Scanlon, “Deep learning at shallow end: Malware classification at non-domain experts,” Proceedings of the Eighteenth Annual DFRWS USA, pp. 3-4, 2018.

[11] Matilda Rhode, Pete Burnap and Kevin Jones, “Early stage malware prediction using recurrent neural networks,” Cryptography and security, Cornell University Library, pp. 3, 2017.

[12] Tom Young, Hazarika D, Poria, S and Cambria E, “Recent trends in deep learning based natural language processing,” CoRR , vol. abs/1708.02709 , 2017. [Online]

[13] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, “An approach for detection and family classification of malware based on behavioral analysis,” International Conference on Computing, Networking and Communications (ICNC), pp. 1–5, 2016

[14] B. Kolosnjaji, A. Zarras, G. Webster, C. Eckert and Q. Bai, “Deep Learning for Classification of Malware System Call Sequences,” Springer International Publishing, pp. 137–149, 2016

[15] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba and T. Yada, “Efficient dynamic malware analysis based on network behaviour using deep learning,” IEEE Global Communications Conference (GLOBECOM), pp. 1–7, 2016