Malware detection approach based on deep convolutional neural networks

Hoda El Merabet* and
Abderrahmane Hajraoui

Department of Physics,
Faculty of Science,
Abdelmalek Essaadi University,
Tetuan, Morocco
Email: helmerabet@uae.ac.ma
Email: hajraouiabder@gmail.com
*Corresponding author

Abstract: Malware detection field becomes more valuable nowadays regarding the continuously growing number of malware codes emerging everyday. Besides, machine learning techniques have been widely used in various fields. For the purpose of employing machine learning in malware detection, an executable file should be represented by its features. Therefore, a dataset of labelled benign and malicious files is considered. Then, the developers extract the appropriate features to their model from each file. These features are displayed as inputs to a machine learning classifier. In previous researches, multiple features and classifiers were adopted in different combinations for a better classification. In this paper, we have been interested to PE header fields' features, and a deep convolutional neural network for classification. We extracted the bytes of the PE header fields' values and fed them to our model as greyscale images. Our model is constituted of 31 consecutive convolutional layers. The model was trained on the train dataset, and finally tested on the test dataset. The results were impressive reaching a test accuracy of 97.85%.

Keywords: convolutional neural networks; residual networks; deep learning; PE features; machine learning; malware detection.

Reference to this paper should be made as follows: El Merabet, H. and Hajraoui, A. (2023) 'Malware detection approach based on deep convolutional neural networks', *Int. J. Information and Computer Security*, Vol. 20, Nos. 1/2, pp.145–157.

Biographical notes: Hoda el Merabet is a Doctor in Computer Security. She worked as a Java developer at TSi Solutions in Enschede, the Netherlands. She was responsible for solutions building for the tourism sector. Then, she worked as a freelancer for developing PHP/Laravel applications. Lastly, she received her PhD degree at Abdelmalek Essaâdi University, Tetuan in the field of Computer Security and Networks.

Abderrahmane Hajraoui is a Professor of the Higher Education at University of Abdelmalek Essaâdi. He works in telecommunication field. He is the Director Thesis in the Physics Department and Manager of Communication and Detection Systems Laboratory, Faculty of Sciences, Abdelmalek Essaâdi University, Tetuan, Morocco. His research areas are signal and image processing, automatics, automation systems, simulation systems, antennas and radiation, microwave devices and intelligent wireless sensors networks.

1 Introduction

Cybercriminals choose weaknesses on the machine they are going to hack. We can think that MAC or Linux machines are more secure, but they can be likewise hacked. However, Windows machines are more used and accordingly more exposed to cybercriminals. For this reason, we are building our model focusing Windows executables. The number of malware codes is on an extended elevation every day. With the revolution of machine learning techniques, malware detection field got more attraction to invest on it. Traditional commercial anti-malware programs rely basically on the signature-based static technique, the technique that would never detect new malicious files as they use novel signatures. The capability of machine learning models to learn from training data, and afterwards to generalise to new data is a major reason to have an expanding number of researches interested in this area. There are a lot of factors that influence the success of a model, namely the types of extracted features, the feature selection techniques used in order to get rid of irrelevant features, and the employed machine learning algorithms as mentioned by El Merabet and Hajraoui (2019) in their survey. In this paper, we are relying on the portable executable PE format, which a file format utilised in machines is running Windows operating systems. PE header fields are extracted and used as input features, and a deep convolutional neural network model CNN constituted of a succession of convolutional layers is used for the classification of a file as malicious or benign.

2 Related work on malware detection

There are various types of features that could be extracted from a file in order to train a machine learning classifier. Static features are attributes extracted from the PE header fields. The PE header consists of information needed by the Windows operating system loader to handle any executable code. It includes the size of the file, the number, size, and entropy of the different sections, the API calls, etc. They can be used in a raw form without any treatment, or subjected to a predefined processing to obtain processed features. Dynamic features form another type of features. These ones are obtained after executing the files on a virtual environment in order not to harm our machines. This execution exhibits the behaviour of the files. Multiple features could be then extracted such as the number of bytes received and sent, the RAM used, the number of write and delete requests, etc.

Malware detection approach based on machine learning has been successfully introduced the last two decades. It can utilise either static features, or dynamic features, or even a combination of both of them.

Dynamic features are strong features illustrating the dynamic behaviour of an executable file. Burnap et al. (2018) executed all the training set samples on a virtual environment in order to extract dynamic features. They chose to gather nine activity metrics from each file. These metrics are: CPU user use, CPU system use, RAM use, SWAP use, received packets, received bytes, sent packets, sent bytes, and the number of running processes. By saving these metrics each second in a time-window of 5 minutes, the researchers collected 300 vectors to represent each file. These vectors were subsequently converted to 300 vectors of x-y coordinates using self organising feature maps (SOFMs). The dataset was constituted of 1,188 PE files; half the files were

malicious and the other half was benign. One half constituted the training set and the other half represented the test dataset. Using logistic regression for classification, they obtained a final accuracy of 86.70%.

API calls and operating system resource instances were utilised to identify API call graphs. These graphs represented the input features for the deep learning model of Xiao et al. (2018). The researchers opted for a feature selection approach in order to reduce the size of the original extracted features. For this purpose, various layers of sparse auto encoders (SAEs) were employed earlier than a decision tree classifier. They had a training dataset of equal numbers of malicious and benign files, with 1,760 as a total number of samples. The researchers did not use new unseen dataset specially for testing; nevertheless they used ten-fold cross-validation technique for training and testing. The achieved detection precision was 98.6%. They did not declare the accuracy rate in their report.

With regard to Xiaofeng et al. (2018), n-gram system call sequences were adopted as features. Two models were utilised in combination. The first model was the long-short-term memory (LSTM). The information gain was used in order to remove redundant sub-sequences, then the LSTM model was fed, followed by a max-pooling layer, then by a logistic regression classifier. The second model was the random forest model. The input features for this latter were API statistical features, which are got from the association of two API calls based on the comparison of the two API call sequences hashes. By combining both models, they got a better accuracy than using each model separately. They got a final test accuracy of 95.7%.

Autoencoders have been likewise used for malware detection by Hardy et al. (2016). They constructed the features for their malware detection model from Windows API calls generated from the PE files. After that, they designed a deep learning model based on stacked auto encoders (SAEs). Hardy et al. (2016) chained autoencoders together; the output of one autoencoder was the input of the next one. They used a train dataset constituted of 22,500 malwares and 22,500 benign files. The test dataset contained 5,000 files, half the files were benign and the other half were malicious. Their best results were obtained using three hidden layers with 100 neurons per each layer. The accuracy on the test data was 95.64%.

Since CNNs have proven high performances in image recognition, they were utilised in multiple malware detection and malware classification researches based on machine learning. Abdelsalam et al. (2018) built a model relying on a dynamic approach. The samples were executed on virtual machines in order to capture performance metrics from the execution process. The raw bytes of each file were represented in an image format, the rows contained the executed processes, and the columns contained the 28 extracted features per process; like the number of read and write requests, the amount of memory swapped out to disk and the CPU usage percent. The chosen machine learning classifier was a CNN model based on two convolutional layers and other layers like maxpooling and dropout. They got an accuracy of 90% on the testing dataset.

Barker et al. (2018) converted the raw bytes of each file using an embedding layer. Through this layer, they wanted to avoid a misinterpretation of the meaning of closer byte values by the machine learning classifier as dependent values in context, which is theoretically not true. The output values of this layer were adopted as input features. They built their model from CNNs and recurrent neural networks (RNNs) followed by a fully connected layer of an artificial neural network ANN. A dataset of 400,000 samples

equally distributed between benign and malicious files was used for training. A distinct testing dataset constituted of 77,349 files was used for the final test phase. They obtained an accuracy of 90.90%.

Sharma et al. (2019) relied on their model on a static approach. They extracted bi-gram assembly features. A list of features has the following form, as given by the authors: [‘pop call’, ‘call test’, ‘test jnz, ‘jnz pop’, ‘pop push’]. Their model was based on a one-dimensional convolutional neural network. It is a CNN based on single dimension input and single dimension filters. Even if this model is computationally a light model, the extraction of assembly features from malware is most of the times a complicated task due to obfuscation techniques. The authors test their detector on two datasets. The first dataset is more general since it is using malware executables from nine malware classes, and the sizes of the files are considerably variable. However, this dataset is not large enough. It is constituted of 1,155 malwares and 608 benign files. 80% of the files were used for training, and 20% for testing. The obtained accuracy on the test part of this dataset was 97.51%. The second dataset contains more samples, but they are all viruses and have almost the same size. The final accuracy on this dataset was 99.2%.

Residual networks are a relatively new category of CNNs. They were initially introduced for image recognition (He et al., 2016). They had a great effect in reducing the vanishing gradient problem encountered in multiple deep learning researches. A first approach to introduce these algorithms to malware detection was performed by El Merabet (2020). PE features were extracted from the files and represented by their byte values, then fed to the model as images. The model was mainly built from a convolutional layer, a batch normalisation layer, a max pooling layer, then two residual blocks; each block constituted of two convolutional layers and a batch normalisation layer. Dropout regularisation was used to reduce overfitting. The dataset was constituted of 600 K samples for training and 200 K samples for testing. The obtained test accuracy was 90.38%.

In this paper, we built a model based on PE features which are static features, and a deep CNN model. While choosing this structure, we were motivated by:

- The success obtained by previous researches that were relying on PE features and/or CNNs. PE features do not allow the visualisation of the exact behaviour of each file such as dynamic features. This can be seen as a limitation. However, they include all the information necessary for the Windows OS to handle the executable code.
- The feasibility of extracting PE features from executable files. Parsing PE headers requires less work contrarily to the dynamic approach which necessitates a long duration of scanning, a feature engineering step done by hand, and a big utilisation of resources.
- The belief that adding more layers and constructing a deeper CNN will get better results.

3 Convolutional neural networks CNNS

3.1 Strengthening vision with CNNs

Convolutional neural networks (CNNs) are a prominent subclass of deep learning algorithms. They require images as inputs. Considering an image, several wasted space can be observed. Each image is constituted of a fixed number of pixels; hence it will be interesting to find a manner to condense it to just the important features that make the difference between one class of images and another. That is the importance of convolutions. That implies introducing a number of filters to pass them through the image like in image processing, and subsequently attain new images called feature maps. Each convolution or the application of each filter in other words, will change the image in such a way that particular features get emphasised.

Together with convolutions, pooling is widely used in CNNs. Pooling is a way of compressing an image; it reduces the size of a feature map almost always by a factor of 2. It takes a region of a fixed number of pixels; typically, 2 * 2, and reduces it into one pixel in the next layer. Depending on the chosen type of pooling, this pixel value can be the maximum value of the pixels in the region for max pooling, the sum of all values for sum pooling, their average for average pooling, etc.

3.2 CNNs for malware detection

The ability of CNNs to learn the existence of a feature regardless of its position, made from them a widely used classification technique in image recognition. Since the different parts of a PE executable file can be placed anywhere within the file, CNNs have been introduced to malware detection field in plentiful researches. By representing the extracted features from a PE file as a matrix, we could convert each file to an image and pass it to our CNN model.

4 Implementation and experimental results

4.1 Dataset used

The dataset used in this study is the EMBER dataset (Anderson and Roth, 2018). It is constituted of 900 K training samples (300 K malicious, 300 K benign, and 300 K unlabelled). We had interest in using only the benign and the malicious ones in our experiments since we are relying on supervised learning. Besides, it consists of 200 K test samples (100 K malicious and 100 K benign). The training of our model was done on 80% of the 600 K training samples. We used the remaining 20% of the data for validation at the end of each epoch. The 200K test samples were hold out to expose them to the built model at the end as previously unseen data, which reflects in more rigor the effectiveness of the model. The utilised features are PE header fields' values. Figure 1 shows an example of a part of PE features representing an executable file as given by Anderson and Roth (2018).

Figure 1 Raw features extracted from a PE file


```
"sha256": "000185977be72c8b007ac347b73ceb1ba3e5e4dae4fe98d4f2ea92250f7f580e",
"appeared": "2017-01",
"label": -1,
"general": {
"file_size": 33334,
"vsize": 45056,
"has_debug": 0,
"exports": 0,
"imports": 41,
"has_relocations": 1,
"has_resources": 0,
"has_signature": 0,
"has_tls": 0,
"symbols": 0
},
"header": {
"coff": {
"timestamp": 1365446976,
"machine": "I386",
"characteristics": [ "LARGE_ADDRESS_AWARE", ..., "EXECUTABLE_IMAGE" ]
},
"optional": {
"subsystem": "WINDOWS_CUI",
"dll_characteristics": [ "DYNAMIC_BASE", ..., "TERMINAL_SERVER_AWARE" ],
"magic": "PE32",
"major_image_version": 1,
"minor_image_version": 2,
"major_linker_version": 11,
"minor_linker_version": 0,
"major_operating_system_version": 6,
"minor_operating_system_version": 0,
"major_subsystem_version": 6,
"minor_subsystem_version": 0,
"sizeof_code": 3584,
"sizeof_headers": 1024,
"sizeof_heap_commit": 4096
}
},
"imports": {
"KERNEL32.dll": [ "GetTickCount" ],
...
},
"exports": []
"section": {
"entry": ".text",
"sections": [
{
"name": ".text",
"size": 3584,
"entropy": 6.368472139761825,
"vsize": 3270,
"props": [ "CNT_CODE", "MEM_EXECUTE", "MEM_READ"]
},
...
]
},
"histogram": [ 3818, 155, ..., 377 ],
"byteentropy": [0, 0, ... 2943 ],
"strings": {
"numstrings": 170,
"avlength": 8.170588235294117,
"printabledist": [ 15, ... 6 ],
"printables": 1389,
"entropy": 6.259255409240723,
"paths": 0,
"urls": 0,
"registry": 0,
"MZ": 1
},
}
```


Source: Anderson and Roth (2018)

4.2 Architecture

Motivated by the success of CNNs in image recognition primarily, and secondly by the great results of these classifiers in the previous malware detection researches, we were convinced that a more examined and tested architecture based on CNNs could give favourable results in the classification of benign and malicious files.

We are experimentally testing the efficacy of our approach on the EMBER dataset described in Section 4.1. The authors of this dataset provided the vectorising of raw features of all samples in their open dataset. Each file is represented as a feature vector of dimension 2,351. A CNN model needs images as inputs. To implement our malware detection model, we chose to represent each sample as a greyscale image of 50 × 50.

Accordingly, we padded each vector by 149 zeros, then we reshaped it to an array of 50 × 50 to feed the model.

Multiple implementations were tested in our experiments. They were based on multiplying the number of convolutional layers. Throughout the experiments, we tried diverse values for the learning rate, the batch size, and the number of filters for each convolutional layer. Our advantageous architecture with 31 consecutive convolutional layers allowed us to reach an accuracy of 97.85%. Ultimately, we retained three perspectives or models. In the first perspective denoted by model1, we used residual blocks with batch normalisation. In the second one designated by model2 we omitted the batch normalisation and kept the residual blocks. Through model3 we mark our third perspective where we omitted both the shortcut connections from the residual blocks and the batch normalisation layers. Model3 is the model that allowed us to reach our higher accuracy rate. After several trials of parameter tuning, the best results were got using a learning rate of 0.00007, a batch size of 128, and 64 filters for the different convolutional layers.

All of our three models follow the same steps at the beginning of the architecture. As mentioned above, the PE header fields' values of each executable file are represented as an image of dimension 50*50, to design the input layer. Then a Gaussian noise layer is added. This layer has a regularising effect and helps in lessening overfitting. Afterwards, a 2D convolutional layer is added with 64 filters. Rectified linear unit (ReLU) is used as activation function. After that, a dropout regularisation layer is included with a factor of 0.2 as another way of reducing overfitting. Then, a max pooling layer is introduced. At this phase, the architecture of the three models will differ. Subsequently, ten residual blocks each one constituted of three convolutional layers and a ReLU activation function are added for model1 and model2, and ten opposite blocks without shortcut connections are implemented for model3. As concerns model1, each convolutional layer inside the residual block is followed by a batch normalisation layer. Afterwards, a new dropout regularisation layer with a factor of 0.2 is added for the three models. Finally, by reason of involving binary decisions, either benign or malicious class, binary cross-entropy is utilised as loss function.

Figure 2, Figure 3, and Figure 4 depict respectively the structures of the residual blocks used in model1 and model2, and the structure of the opposite block without shortcut connection in model3. This latter contains three consecutive convolutional layers that would be directly connected to the next convolutional layer in the following block.

4.3 Results and discussion

All the experiments in this paper were executed on a laptop computer with Intel® Core (TM) i7-6500U @ 2.50 GHz, 2.59 GHz, and 16 GB of RAM. This laptop includes the GPU Intel® HD Graphics 520 for the parallelisable part of the computation. The problem is that we could not exploit this GPU in our model training due to some limitations according to the chosen framework TensorFlow, as described further below.

Figure 2 A residual block in model1 (see online version for colours)

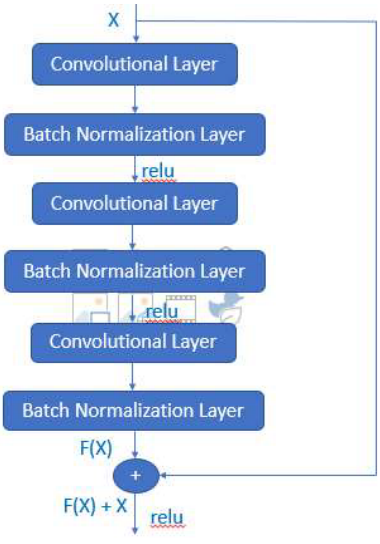
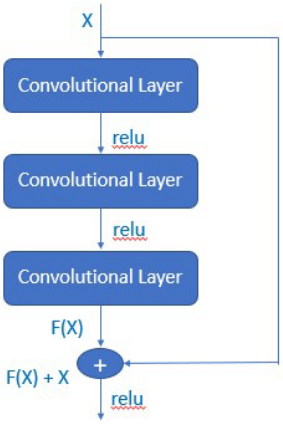


Figure 3 A residual block in model2 (see online version for colours)



Our code is written in Python programming language. For developers of machine learning projects, Python is becoming the dominant preference. One reason for that is the great supply of libraries and frameworks that simplify coding and reduce development time. We chose TensorFlow as framework. TensorFlow GPU support requires the Nvidia Graphics Card (https://www.tensorflow.org). This means that for an increased GPU

accelerated training, a dedicated GPU is needed, and the Intel onboard graphics card in not suitable for that purpose. For this reason, we were relying only on CPU for our model building.

Model1 gives an idea about the improvement of the results. It got an accuracy of 87.66%. Figure 5 represents the train and validation loss and accuracy obtained by this model.

Figure 4 The opposite block in model3 (see online version for colours)

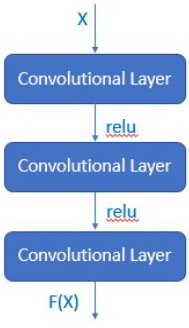
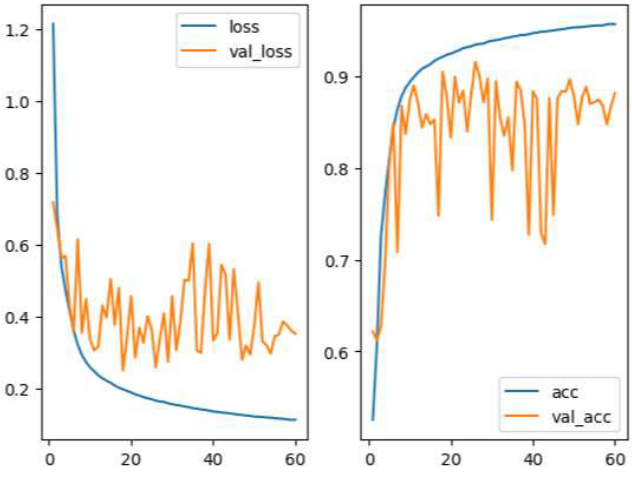


Figure 5 Train and validation accuracy and loss obtained with model1 (see online version for colours)

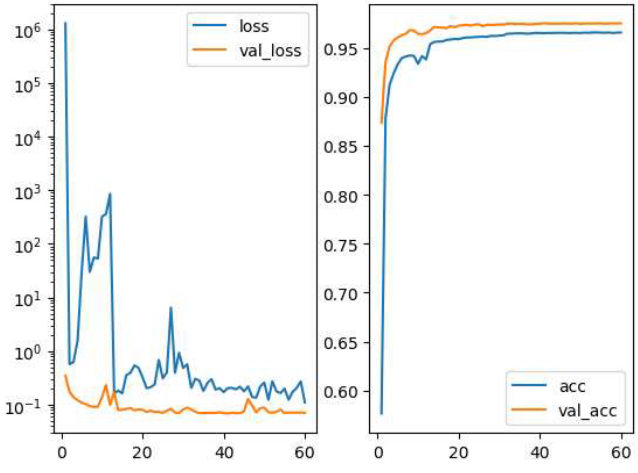


This accuracy rate is worse than the one got by the two-block residual network model built by El Merabet (2020), namely 90.38%, even though we multiplied the number of residual blocks from two to ten blocks. Both models use batch normalisation layers.

In what concerns model2; it got a final accuracy of 96.61%. The best validation accuracy, namely 97.53%, was obtained at epoch 60 which represents the end of the training. The test of this saved model on the previously unseen samples of the test dataset gave an accuracy of 96.61%. This result represents a decrease of 0.92% from validation to test data. As mentioned in the research of El Merabet and Hajraoui (2019), this decrease is a normal behaviour of machine learning studies while confronting the model

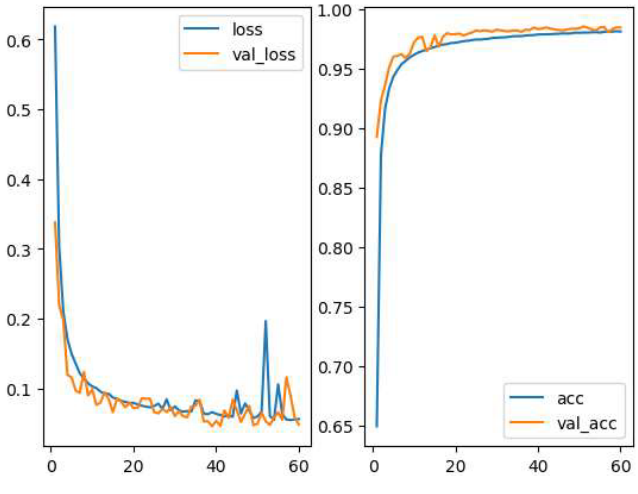
to new data. Figure 6 represents the train and validation loss and accuracy obtained by model2.

Figure 6 Train and validation loss and accuracy obtained with model2 (see online version for colours)



As regards model3, it obtained a best validation accuracy of 98.48% at epoch 42. The number of our training epochs is 60, thus between the epochs 43 and 60, the model continued to learn peculiarities of the training data, and became no more able to accurately generalise to new unseen data, this is the phenomenon known by overfitting. In that case we opted for an early stopping approach, where we took the model saved at epoch 42. This allowed us to get a final model accuracy on the test data of 97.85%. This value represents a decrease of 0.63%. Figure 7 represents the train and validation loss and accuracy obtained with model3.

Figure 7 Train and validation accuracy and loss obtained by model3 (see online version for colours)



The decrease proportion of 0.63% from validation to test data in model3 is absolutely a tiny value compared to previous researches of other authors. For instance, the accuracy of the model of Lo et al. (2016) constituted of a combination of SVM and ANN showed a decrease of 1.20% from validation to test data. As for Burnap et al. (2018), their ANN model showed a decrease of 2.45% by testing on previously unseen data. In what concerns Abdelsalam et al. (2018), their CNN model registered a lessening of 7% in accuracy. Finally, with respect to El Merabet (2020), the two-block residual network marked a decrease of 1.91%.

For our two models without batch normalisation, namely model2 and model3, we noticed that throughout the whole training, the train accuracy was almost all the time less than the validation accuracy and the train loss was higher than the validation loss, this is clear in Figures 6 and 7. Validation data is a portion of data that the model does not use to learn from in its training phase, but it is exposed to the model at the end of an epoch. At the beginning of each epoch, the model takes a new portion from the training data and keeps it to the end of the epoch as validation data. Based on this data, validation accuracy is calculated to reflect the improvement of the model from epoch to epoch. For this reason, the validation accuracy should ordinarily be less than the training accuracy. This was not the case with our models. It can be due to the Gaussian noise layer added at the beginning and the discarding of batch normalisation layers.

The accuracy rates of the models explained in this paper are shown in Table 1.

Table 1 Test accuracies on EMBER test dataset for our models

	Accuracy
Model1	87.66%
Model2	96.61%
Model3	97.85%

From Table 1, we can see clearly that the residual network model model2 does not perform better than the opposite plain model model3. The plain model performs better with a larger progress of 1.24%. The omitting of batch normalisation layers had a great effect in the results' improvement.

The impressive gain in accuracy from a model with two residual blocks (El Merabet, 2020) to a model with ten residual blocks (our present model) is due to adding more layers plus omitting the batch normalisation step. According to Ioffe and Szegedy (2015), the procedure of batch normalisation accelerates the training of ANNs and also has a regularising effect. In our study, this was not the case. We tested plenty deep CNN models with batch normalisation, and all of them got accuracy rates smaller than the same models by omitting batch normalisation. Besides, the training was not accelerated, but slowed down instead, as represented in Table 2. The considerable contribution in our results was due to the exclusion of the batch normalisation step.

Table 2 Training time per epoch in seconds (s)

	Training time
Model1	15,296
Model2	14,703
Model3	14,557

Table 2 shows that our models are relatively time consuming. This remark is deductible as the models are too deep. Forthwith, if we compare our three current models, we conclude that the batch normalisation layers add more time to the training, and the residual blocks yet much more time.

While executing our experiments, we had the expectations that the residual network model model2 will perform better than the plain model Model3, yet it was the opposite. Model3 constituted of 31 consecutive convolutional layers performs better than model2, giving us an accuracy of 97.85%. This accuracy is even better than other previous researches as shown in Table 3.

Table 3 Comparison of our deep CNN model with previous researches

Research	*Model structure*	*Accuracy*
Abdelsalam et al. (2018)	• Machine activity metrics • CNN network	90%
El Merabet (2020)	• PE features • Residual networks	90.38%
Barker et al. (2018)	• Raw byte sequences • CNN + RNN	90.90%
Burnap et al. (2018)	• Machine activity metrics • Logistic regression	94.60%
Hardy et al. (2016)	• API calls • Stacked AutoEncoders	95.64%
Masud et al. (2007)	• Binary and assembly ngrams + DLL function calls • SVM	96.30%
Sharma et al. (2019)	• Bi-gram assembly instructions • One-dimensional CNN	97.51%
Our deep CNN model	• PE features • Deep CNN	*97.85%*

The researches cited in Table 3 use different datasets, except our research and the one of El Merabet (2020), both of them use the EMBER dataset (Anderson and Roth, 2018). Furthermore, each research is based on different input features and classification algorithms, as mentioned on the table.

5 Conclusions

CNNs have been used in malware detection field by providing several approaches. The new model built in this paper is based on CNNs using 31 consecutive convolutional layers without batch normalisation. We have been interested to deep convolutional neural networks with a new architecture. In our approach, we have taken the portable executable header fields values, converted them to byte values, and subsequently fed them to our deep convolutional neural network model as greyscale images. The results as shown above are impressive reaching an accuracy of 97.85%. That can be much more improved

in the future by more hyper-parameter tuning or by choosing other types of features like machine activity metrics that could very well train other classification models. The results can be further enhanced by the use of feature selection techniques in order to get rid of insignificant features, by training and testing on additional datasets, or by implementing deeper ResNets with greater number of hidden layers. Nevertheless, the training of deeper networks needs integrating GPUs for more powerful computations.

References

Abdelsalam, M., Krishnan, R., Huang, Y. and Sandhu, R. (2018) 'Malware detection in cloud infrastructures using convolutional neural networks', in *Proceedings of the 11th IEEE Conference on Cloud Computing (CLOUD)*, San Francisco, CA, July 2–7, pp.162–169.

Anderson, H. and Roth, P. (2018) *EMBER: An Open Dataset for Training Static PE Malware Machine Learning Model*, April, ArXiv e-prints. arXiv: 1804.04637.

Burnap, P., French, R., Turner, F. and Jones, K. (2018) 'Malware classification using self-organizing feature maps and machine activity data', *Computers and Security Journal*, March, Vol. 73, pp.399–410.

El Merabet, H. (2020) 'A first approach to malware detection using residual networks', *International Journal of Computer Science, Communication and Information Technology CSCIT*, February, Vol. 9, No. 1, pp.1–6.

El Merabet, H. and Hajraoui, A. (2019) 'A survey of malware detection techniques based on machine learning', *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 10, No. 1, pp.366–373.

Hardy, W., Chen, L., Hou, S., Ye, Y., and Li, X. (2016) 'DL 4 MD: a deep learning framework for Intelligent malware detection', in *Proceedings Int'l Conf. Data Mining, DMIN'16*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016) 'Deep residual learning for image recognition', Paper presented at the *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Las Vegas, NV, June, pp.770–778.

Ioffe, S. and Szegedy, C. (2015) 'Batch normalization: accelerating deep network training by reducing internal covariate shift', in *CoRR*, arXiv: 1502.03167.

Lo, C.T.D., Pablo, O. and Carlos, C.M. (2016) 'Towards an effective and efficient malware detection system', Paper presented at the *IEEE International Conference on Big Data*, Washington, DC, USA, December.

Masud, M.M., Khan, L. and Thuraisingham, B. (2007) 'A hybrid model to detect malicious executables', Paper presented at the *IEEE International Conference on Communications, ICC 2007*, Glasgow, Scotland, 24–28 June, pp.1443–1448.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B. and Nicholas, C (2018) 'Malware detection by eating a whole exe', in *AAAI Workshop on Artificial Intelligence for Cyber Security*, arxiv: 1710.09435

Sharma, A., Malacaria, P. and Khouzani, M. (2019) 'Malware detection using 1-dimensional convolutional neural networks', *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Stockholm, Sweden, pp.247–256, DOI: 10.1109/ EuroSPW.2019.00034.

Xiao, F., Lin, Z., Sun, Y. and Ma, Y. (2019) 'Malware detection based on deep learning behaviour graphs', *Journal of Mathematical Problems in Engineering*, February, Vol. 2019, pp.1–10.

Xiaofeng, L., Xiao, Z., Fangshuo, J., Shengwei, Y. and Jing, S. (2018) 'ASSCA: API based sequence and statistics features combined malware detection architecture', in *Procedia Computer Science*, January, Vol. 129, pp.248–256.