Harnessing Performance Counters to Detect Malware Using Deep Learning Models

Omar Mohamed
Necmettin Erbakan University, Konya, Turkey
University Politehnica of Timişoara, Timisoara, Romania
omar@omarmohamed.com

Ciprian-Bogdan Chirila
Department of Computers and Information Technology
University Politehnica of Timişoara
Timişoara, Romania
chirila@cs.upt.ro

Abstract—**Computing systems are challenged by security exploits and malware. The following methods are used for detecting anomalies and discovering vulnerabilities in computing systems: malware aware processors, static program analysis, and dynamic program analysis. Online hardware to detect malware is not always a practical and scalable solution because of the costs. Automated static analysis tools have limited performance and detection capabilities that may not meet the criticality requirements of the project regarding static analysis methods. In the latest trends, dynamic analysis has overcome static analysis. Several approaches have been used to analyze performance counters in this sense. Performance counters are collected from both operating systems/software and processors/hardware and stored as time series: 1) in the presence and 2) in the absence of malware. For software performance counters (SPCs), fourteen deep learning models were used for time series classification, while for hardware perfornamce counters (HPCs), ten deep learning models were used. For SPCs two models were able to detect accurately malware in infected operating systems, while the rest tend to overfit the data. For HPCs three models were able to detect malware.**

Index Terms—**malware, software performance counters, hardware performance counters, program behaviour, time series classification, deep learning classification models, recurrent neural networks**

I. Introduction

In the context of globalization the physical border security devices of a state-nation do not protect its digital infrastructure. Nowadays security attacks occur at a high rate and are various and complex. It was estimated by anti-virus companies that the number of malware is at the size of tens of millions. Malware has a huge rate of growth, namely over 300 new threats are created each minute. The malware term denotes malicious software that can damage other software systems.

Companies and state-nation organizations servers are protected by hardware e.g. firewalls, intrusion prevention systems (IPS), intrusion detection systems (IDS) and software e.g. antiviruses (AV) solutions.

Hardware protection is usually expensive and thus not always accessible.

The problem with AVs is that in the traditional approach they have static signatures in order to detect malware. Attackers can program malware such that it exhibits benign software signatures. On the other hand, AVs are prone to exploits and they are consuming a considerable amount of resources, so they are difficult to be used in real-time protection.

One solution is a detector that classifies programs by identifying malware and then helps applying expensive software based solutions. Such a classification can be made based on logistic regression and neural networks. Hardware classifiers implemented by FPGA malware aware processors are not always practical and scalable. HPCs do not require additional hardware and relies on the presence of the microprocessor. One disadvantage of the HPCs is that their collection in a continuous time series is limited to 4-6 event types.

In this context a malware detector based on the dynamic behavior of computer programs may overcome the static based detection. The dynamic behavior is expressed as performance counters time series collected from the operating system and/or from the processor.

In this paper we augment a framework for training and evaluating deep learning models that detects malware based on SPCs time series classification [13].

In Figure 1 we present conceptually our approach.

We start our approach at a Virtual Machine (VM) level running programs in the presence and absence of malware. The malware used in the experiments were obtained from the VirusTotal [18] company. On the VM we run in parallel a tool that collects 23 performance counters grouped into time series. Next, the results are normalized using statistical operators like Min-Max and Z-Score. Additionally, in our study we use time series obtained from other experiments based on HPCs [22]. On the normalized time series we train fourteen classification models for SPCs and ten classification models for HPCs [7]:

1) Residual neural network (ResNet);
2) Convolutional Neural Network (CNN);
3) Time Le-Net (t-LeNet);
4) Fully Convolutional Neural Network (FCN);
5) Multi Layer Perceptron (MLP);
6) Encoder;
7) Multi-scale Convolutional Neural Network (MCNN);
8) Multi Channel Deep Convolutional Neural Network (MCDCNN);
9) Time Convolutional Neural Network (Time-CNN);
10) Inception Time.
11) RNN implementaions (LSTM, GRU and vanilla RNN) [1]

On the trained classification models accuracy tests were performed and corresponding graphs were plotted. We consider

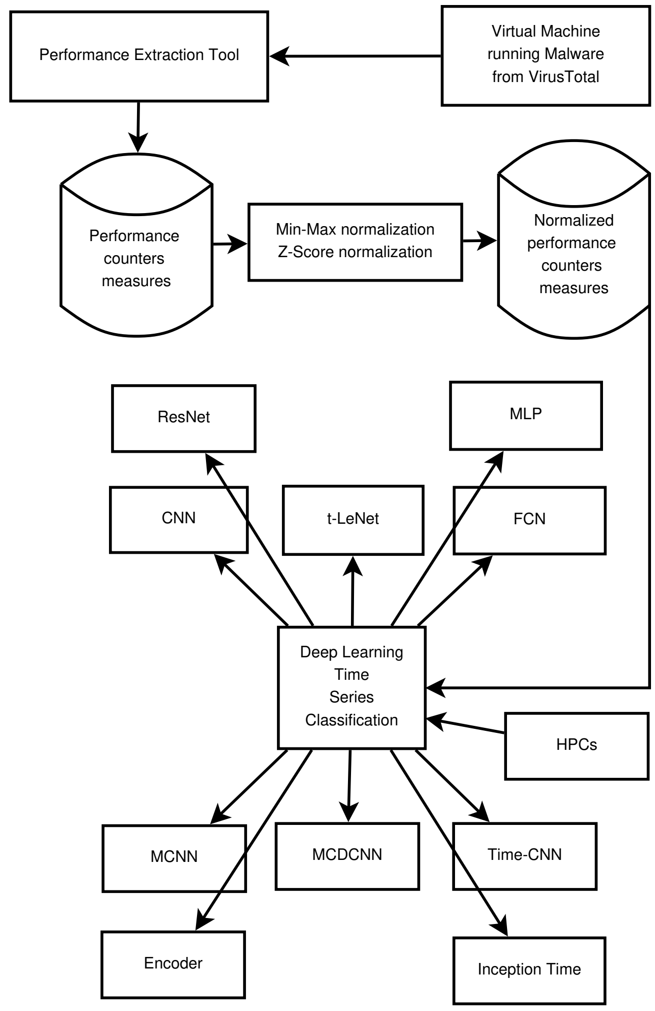
Performance Extraction Tool
Virtual Machine running Malware from VirusTotal
Performance counters measures
Min-Max normalization Z-Score normalization
Normalized performance counters measures
ResNet
MLP
CNN
t-LeNet
FCN
Deep Learning Time Series Classification
HPCs
MCNN
MCDCNN
Time-CNN
Encoder
Inception Time

Fig. 1. Approach

that the trained classifiers could be used on a real machine as a malware detection tool.

The paper is structured as follows. Section II presents related works in the field of program behavior analysis based on performance counters and deep learning classification models. Section III presents the design of the performance counters extraction tool and the configuration of the deep learning classification models. Section IV-A presents the experimental results from the trained classification models. Section V analyzes the experimental results. Section VI concludes and sets the future work.

II. Related Works

The works of [9], [13] are the seminal papers for our approach. The work of Kadiyala et al. [9] explore performance counters to analyze the behavior of programs at run-time. They developed a prototype bases on operating system's calls to capture the values of performance counters. Thus, time series are formed describing the behavior of programs in a given period of time. Next, they develop a semi-supervised clustering algorithms to group programs by their intrinsic behavior. The experiments they carry are based on 18 programs grouped in 4 clusters: web browsers, text editors, image viewers and audio players. They conclude from the experiments that the performance counters accurately differentiate the dynamic behavior of four clusters of programs. They claim that the results are not influenced by the virtual or physical environment where the programs run.

In [11] is presented a machine learning based approach that optimizes hyper-parameters of machine learning models applied to malware detection problems. Two automated machine learning (AutoML) frameworks are used, namely AutoGluon-Tabular and Microsoft Neural Network Intelligence. They optimize the parameters of Light Gradient Boosted Machine (LightGBM) that is in charge with malware samples classification. In our approach we use classification models with no hyper-parameter optimization from [7] except two models.

In [4] is presented a hardware-assisted malware detection (HMD) technique using machine learning classifiers. The approach is based on low-level micro-architectural events captured by hardware performance counters. In the approach they create an adversarial attack on the HMD systems using an adversarial sample predictor. Their intention is to tamper the security by introducing perturbations in the HPC series.

In [17] is presented a machine learning approach based on hardware assisted profiling of browser code in real-time. The model classifies unauthorized mining applications even if their code is heavily obfuscated or encrypted.

In [15] is presented a machine learning approach that detects malware based on HPCs. They implemented the classifier at the OS kernel level which slowed down the system, which, finally they accelerated on an FPGA. They used the following models: Multi Layer Perceptron (MLP) and OneR.

In [14] is presented a deep semi-supervised learning-based network anomaly detector operating in heterogeneous information systems. They rely on deep recurrent autoencoder which learns the time series of normal network behavior and detects network anomalies. Optimizations are applied on the proposed features reducing their number by 94% without loosing accuracy.

In [2] is presented a mathematical framework to investigate the probability of malware detection based on HPC monitoring at a fixed sample interval. HPCs are read after every 500 cycles. They use a control flow graph representation of programs. The graph visit has a set of HPCs values as signatures for each visited node.

In [22] is presented an approach of detecting malware based on HPCs using models like: Decision Trees, Random Forest, K-Nearest Neighbours (KNN), Adaboost, Neural Net (NN) and Naive Bayes. They cross-validated their model 1000 times which resulted in low F1-scores. They use a sampling frequency of 1 KHz for their HPC measuring tool. They downloaded 1000 malware from VirusTotal, grouped in 35 distinct malware families and run them for 1 minute.

III. Experimental Setup

In this section we will present our experimental setup based on: i) a C# tool `PerfExtract` for performance counters extraction as time series; ii) runtime environment for malware and benignware; iii) 14 for SPCs and 10 for HPCs deep learning classification models used to analyze the time series; iv) a Python prototype for training the models and plotting the results.

A. Software Performance Counters Extraction Tool - PerfExtract

The `PerfExtract` tool was developed in C# as an application that extracts the following counters from the Microsoft™Windows operating system:

1) % Privileged Time - percentage of non-idle processor spent executing code in privileged mode;
2) Handle Count - the number of handles currently opened by a process;
3) IO Read Operations/sec - the rate of reads per second for files, network and I/O devices;
4) IO Data Operations/sec - the rate of reads and writes per second for files, network and I/O devices;
5) IO Write Operations/sec - the rate of writes per second for files, network and I/O devices;
6) IO Other Operations/sec - the rate of other I/O operations for files, network and I/O devices;
7) IO Read Bytes/sec - the rate of bytes read per second for files, network and I/O devices;
8) IO Write Bytes/sec - the rate of bytes issued to I/O operations per second for files, network and I/O devices;
9) IO Data Bytes/sec - the rate of bytes read and wrote I/O operations per second for files, network and I/O devices;
10) IO Other Bytes/sec - the rate of bytes used in control operations per second for files, network and I/O devices;
11) Page Faults/sec - the rate of page faults per second handled by the processor. It includes both type of page faults: disk page faults and memory page faults;
12) Page File Bytes Peak - the maximum amount of virtual memory, in bytes, reserved by the target process to be used for paging files;
13) Page File Bytes - the current amount of virtual memory, in bytes, reserved by the target process to be used for paging files;
14) Pool Paged Bytes - number of bytes from the area of system memory that can be written to the disk;
15) Pool Non-paged Bytes - number of bytes from the area of system memory that can not be written to the disk;
16) Private Bytes - the size in bytes of the memory allocated that can not be shared with other processes;
17) Priority Base - the current priority base for the target process;
18) Thread Count - the number of threads that are running in the target process;
19) Virtual Bytes Peak - the maximum size in bytes for the virtual address space used by the target process;
20) Virtual Bytes - the size in bytes for the virtual address space used by the target process;
21) Working Set Peak - the maximum size in bytes for the working memory set of the target process;
22) Working Set - the size in bytes of the working memory set for the target process;
23) Working Set Private - subset of working set reflecting the un-shared amount of memory.

B. Malware and Benignware Runtime Environment

In the experiment we used a virtual machine on a cloud as we have to execute the malware. Such an operation is definitely harmful to run on the working computer.

An Elastic Computing Cloud (EC2) from Amazon Web Services (AWS) [5] was be used for experimenting. We kept the machine isolated from the Internet, only a specific IP was allowed through the Internet firewall.

AWS is suitable for testing ransomware which executes normally even without internet connection. In our experiment an Internet connection is required as most malware will not run without connection. So, VirtualBox [6] was used to create a virtual machine. The machine has installed Microsoft™Windows 7 with all required drivers and run-time environments.

The time frame for the extraction was set at 30 seconds. The reason for this decision is based on the fact that programs may be used in different scenarios by different users. The startup period is more likely to be the same for most usage scenarios. The acquisition rate was set at 0.5 seconds in order to capture decent length time series.

The PerfExtract tool may be executed using the following syntax:

```
perfextract (-f <path to exe file> |
-p <pid of specific running program>)
[options] [-o <path to output folder>]
-c Track child processes
```

The -f option instructs the tool to run the executable image file as an argument and record the performance counters. The -p option instructs the tool to track the process by its identifier and record its performance counters. The -c option instructs the tool to track the child processes of the target process. The -o option configures the output of the tool as a CSV file containing the time series for each performance counter.

1) Malware Experiment: For the malware based experiment we used 409 malware from VirusTotal [18]. We sampled the performance counters at 0.5 seconds and we collected them for a time frame of 30 seconds. We obtained a result of 60 rows. The launch of the `PerfExtract` tool was executed by a Python script applying the `-f` option of the tool specifying the file path of the malware executable image. Some malware created child processes so we used the `-c` option to track their children processes.

2) Benign Applications Experiment: For the benign applications experiment we used 288 benign (safe) applications. The applications types are: browsers, IDEs, background Windows processes etc. To launch the `PerfExtract` tool we used the `-p` option to refer to the PID (process identifier) of the running benign applications.

3) Principal Component Analysis: On the resulted data we perform two PCA analysis to determine the most important counters of our approach. We used the Python SciKit-Learn library in this sense. We concatenated the counter series from all 409 malware and 288 benign processes and labeled them as "malign" or "benign". Next, we performed a standard scaling on the input data using the StandardScaler() class which performs a Z-score normalization. We set the number of components to 2 to facilitate a bi-dimensional representation. As a result we learned that the first principal component contains 28.52% of the variance, while the second principal component contains 17.79% of the variance. Together the two components contain 46.31% of the information.

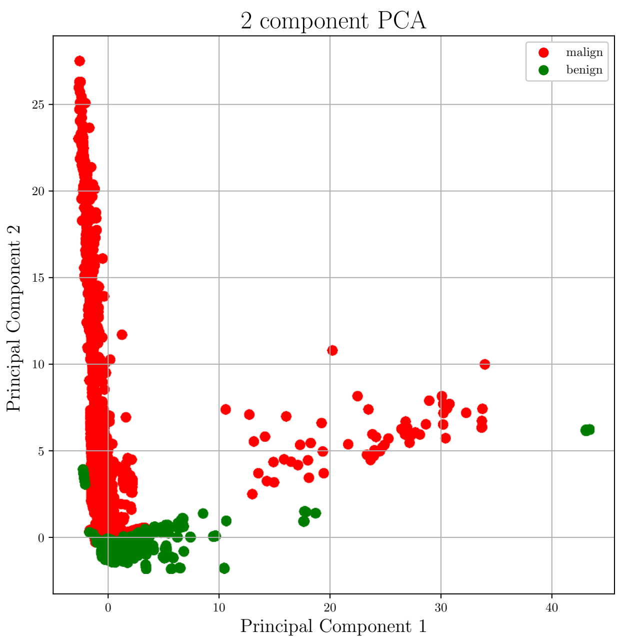

Fig. 2. Principal component analysis for a 2 dimensional space

In Figure 2 we see the result of converting the 23 dimensional space into 2 dimensional space. To be noticed on the graph that the classes do not seem well separated from each other.

```
[0.28524166 0.17793485 0.112478
 0.08549196 0.06712712 0.04766427
 0.04675847 0.03922873 0.03145469
 0.02566163 0.02251158 0.02142496]
```

Fig. 3. Principal component analysis for 95% of the variance

On the other hand, if we perform a PCA analysis with a target of 95% of the variance, we obtain 12 relevant dimensions out of 23 (see Figure 3).

C. Data Preprocessing

The data is processed by several Python libraries such as pandas, sklearn and numpy. Each data sample dimensionality is 60 x 23, so after loading the whole data set we get a N x 60 x 23 numpy array. The data is normalized using min-max normalization:

$$X = \frac{X - X_{min}}{X_{max} - X_{min}}$$

or Z-score normalization:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation.

Finally, we split the data set into: train, validation and test sets.

D. Deep Learning Classification Models

a) Residual Neural Network (ResNet): The network [19] is composed of three residual blocks followed by a Global Average Pooling (GAP) layer and a final Softmax layer.

A shortcut residual connection between convolutional layers is used to enable the flow of the gradient directly through these connections, which reduces the vanishing gradient effect and makes training a deep neural network easier.

The classifier number of neurons is equal to the number of classes in a data set. Each residual block is first composed of three convolutions whose output is added to the residual block's input and then fed to the next layer.

The number of filters for all convolutions is fixed to 64, with the ReLU (Rectified Linear Unit) activation function that is preceded by a batch normalization operation. In each residual block, the filter's length is set to 8, 5 and 3.

The layers (except the final one) in the ResNet architecture have an invariant number of parameters across different datasets, thus we can pre-train a model on any dataset then transfer and fine-tune it on a target dataset without modifying the hidden layers of the model.

b) Convolutional Neural Network (CNN): We use a normal Convolutional Neural Network [10] with 2 convolution layers and 2 average pooling layers followed by a dense layer with the size of classes' count.

The convolution layers has a kernel size of 7 and filter sizes of 6 and 12.

c) Time Le-Net (t-LeNet): This model [12] can be considered as a traditional CNN with two convolutions followed by a fully connected (FC) layer and a final Softmax layer.

There are two main differences with the FCNs: (1) an FC layer and (2) local maximum pooling operations. For both convolutions, the ReLU activation function is used with a filter length equal to 5.

For the first convolution, 5 filters are used and followed by a max pooling of length equal to 2.

The second convolution uses 20 filters followed by a max pooling of length equal to 4.

The convolutional blocks are followed by a non-linear fully connected layer which is composed of 500 neurons, each one using the ReLU activation function. Finally, we use a Softmax classifier.

Due to the use of fully connected layers instead of a Global Average Pooling layers this model doesn't contain much invariant layers, thus the number of parameters needed to be trained increases which limits the transferability of the model to the first two convolutional blocks whose number of parameters depends on the number and length of the chosen filters.

This model adopts two different data augmentation techniques: Window Slicing (WS) and Window Warping (WW) to prevent overfitting especially for the relatively small time series data sets. WW method squeezes or dilates the time series, while WS method ensures that subsequences of the same length are extracted for training the network.

d) Fully Convolutional Neural Network (FCN): FCNs are mainly convolutional networks [19] that do not contain any local pooling layers which means that the length of time series is kept unchanged throughout the convolutions. The model was first proposed in Wang et al. [19] for classifying univariate time series datasets.

The model is composed of three convolutional blocks where each block contains convolution followed by a batch normalization and ReLU activation function. The result of the last block if fed to a Global Average Pooling layer. Finally, a softmax classifier is connected to the GAP layer's output.

All convolutions has a stride of 1 and 0 padding to keep the original length of the time series. The first convolution has 128 filters and filter length of 8, the second convolution has 256 filters with a filter length of 5, and the last convolution has 128 filters with filter length of 3.

In addition, one of the main characteristics of this architecture is the replacement of the traditional final FC layer with a GAP layer, which reduces drastically the number of parameters in a neural network, while enabling the use of the Class Activation Maps (CAM) that highlights which parts of the input time series contributed the most to a certain classification.

One of the advantages of this model is the invariance in the number of parameters across time series of different lengths which enables using transfer learning by pre-training the model on a source dataset then fine-tuning it on the target dataset.

e) Multi Layer Perceptron (MLP): The network [19] contains 4 layers in total where each one is fully connected to the output of its previous layer.

The final layer is a Softmax classifier, which is fully connected to its previous layer's output and contains a number of neurons equal to the number of classes in the data set.

All three hidden FC layers are composed of 500 neurons with ReLU as the activation function. Each layer is preceded by a dropout operation with a rate equal to 0.1, 0.2, 0.2 and 0.3 for respectively the first, second, third and fourth layer.

Dropout is one form of regularization that helps in preventing overfitting. The dropout rate indicates the percentage of neurons that are deactivated (set to zero) in a feed forward pass during training.

This model doesn't have any layer with an invariant number of parameters which makes transfer learning impossible as the number of parameters depends directly on the length of the time series.

f) Encoder: Encoder [16] is a hybrid deep CNN whose architecture is inspired by FCN with a main difference, namely, the GAP layer is replaced with an attention layer. Similarly to FCN, the first three layers are convolutional with some relatively small modifications.

The first convolution is composed of 128 filters of length 5; the second convolution is composed of 256 filters of length 11; the third convolution is composed of 512 filters of length 21.

Each convolution is followed by an instance normalization operation whose output is fed to the PReLU (Parametric Rectified Linear Unit) activation function.

The output of PReLU is followed by a dropout operation and a final max pooling of length 2.

The third convolutional layer is fed to an attention mechanism that enables the network to learn which parts of the time series are important for a certain classification.

Finally, a traditional Softmax classifier is fully connected to the latter layer with a number of neurons equal to the number of classes in the data set.

In addition to replacing the GAP layer with the attention layer, Encoder differs from FCN in three main core changes: (1) the PReLU activation function where an additional parameter is added for each filter to enable learning the slope of the function, (2) the dropout regularization technique and (3) the max pooling operation.

g) Multi-scale Convolutional Neural Network (MCNN): The MCNN architecture [3] is very similar to a traditional CNN model: with two convolutions (and maximum pooling) followed by an FC layer and a final Softmax layer.

This approach is very complex with its heavy data preprocessing step, Window Slicing (WS) method as a data augmentation technique. WS slides a window over the input time series and extract subsequences, thus training the network on the extracted subsequences instead of the raw input time series. Following the extraction of a subsequence from an input time series using the WS method, a transformation stage is used.

The output of each convolution in the first convolutional stage is concatenated to form the input of the subsequent convolutional layer. Following this second layer, an FC layer is deployed with 256 neurons using the sigmoid activation function. Finally, the usual softmax classifier is used with a number of neurons equal to the number of classes in the dataset.

h) Multi Channel Deep Convolutional Neural Network (MCDCNN): The architecture [21] is mainly a traditional deep CNN with one modification for MTS data: the convolutions are applied independently (in parallel) on each dimension (or channel) of the input MTS.

Each dimension for an input MTS will go through two convolutional stages with 8 filters of length 5 with ReLU as the activation function.

Each convolution is followed by a max pooling operation of length 2.

The output of the second convolutional stage for all dimensions is concatenated over the channels axis and then fed to an FC layer with 732 neurons with ReLU as the activation function.

Finally, the Softmax classifier is used with a number of neurons equal to the number of classes in the data set.

By using an FC layer before the softmax classifier, the transferability of this network is limited to the first and second convolutional layers.

i) Time Convolutional Neural Network (Time-CNN): Time-CNN approach was originally proposed by Zhao et al. (2017) for both univariate [20] and multivariate TSC.

The first characteristic of Time-CNN [20] is the use of the mean squared error (MSE) instead of the traditional categorical cross-entropy loss function. Another difference to traditional CNNs is the use of a local average pooling operation instead of local max pooling. In addition,a one convolution for all the dimensions of a multivariate is applied for classification task. The final classifier is fully connected directly to the output of the second convolution, which removes completely the Global Average Pooling layer without replacing it with a fully connected non-linear layer.

The network is composed of two consecutive convolutional layers with respectively 6 and 12 filters followed by a local average pooling operation of length 3.

The convolutions adopt the sigmoid as the activation function.

The network's output consists of an FC layer with a number of neurons equal to the number of classes in the data set.

j) Inception Time: This classifier [8] is built of 2 residual blocks, each block contains 3 inception modules instead of fully connected layers.

Each block's output is transferred using a shortcut layer to the next block's input to mitigate the vanishing gradient problem by allowing a direct flow of the gradient.

After the residual blocks a Global Average Layer in used and finally a fully connected layer with Softmax function and a number of neurons equal to class the number of classes.

k) RNN: LSTM, GRU, and simple RNN classifiers implemented by Shekoofeh Azizi [1]

IV. Experimental Results

A. Software Performance Counters

In this section we will present the experimental results obtained for each of the deep learning classification models.

By looking at Figures 6, 7, 11 and 10 we can see that these models tend to be overfitting, especially t-LeNet (see Figure 11) model.

The MCNN model failed with 58% accuracy, 50% recall, 29% precision and 0.37 F1 score.

MLP (see Figure 9) and CNN (see Figure 4) models overfitted with 100% accuracy, and precision, recall, and a F1 score of 1.

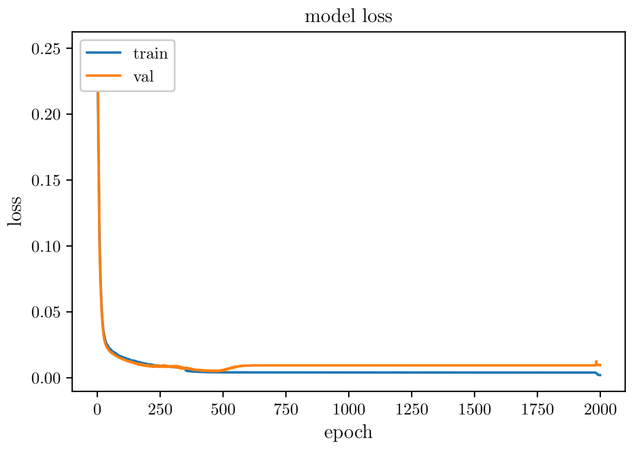

Fig. 4. SPC CNN

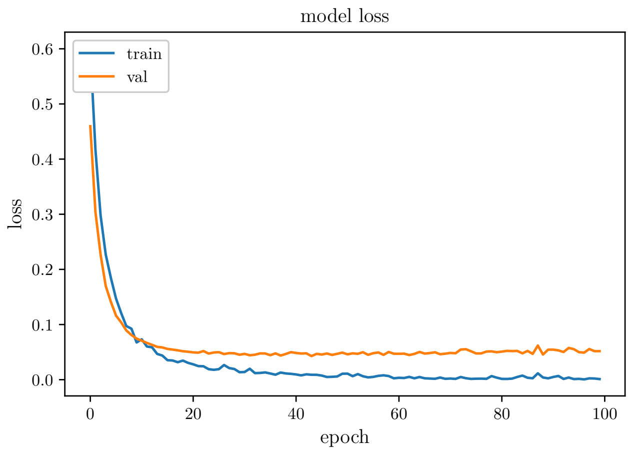

Fig. 5. SPC Encoder

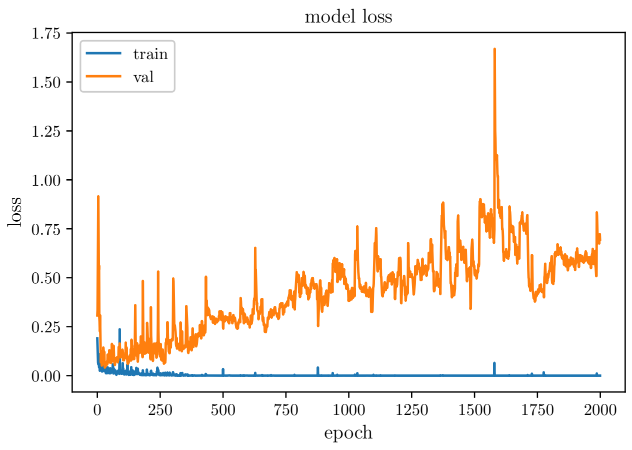

Fig. 6. SPC FCN

Encoder and MCDCNN (see Figure 8) models performed well even with our small data set with a 98% accuracy, 97% recall and precision, and 0.97 F1 score.

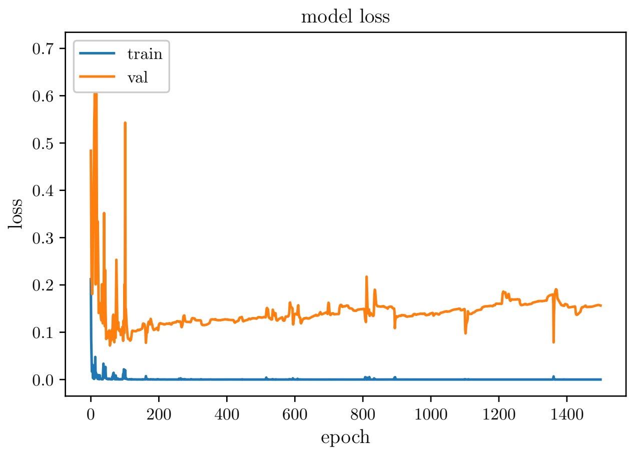

Fig. 7. SPC Inception Time

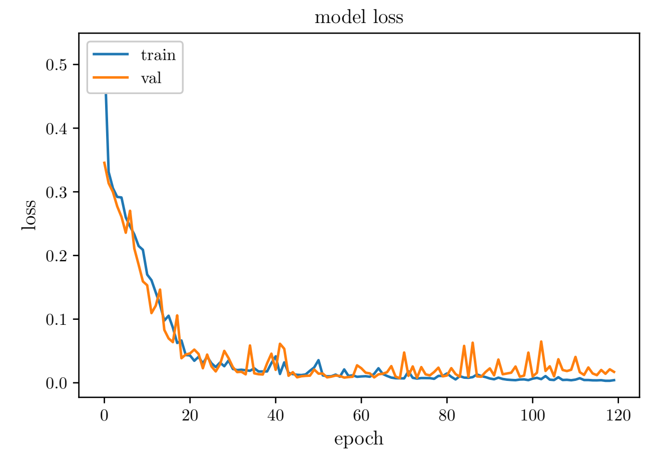

Fig. 8. SPC MCDCNN

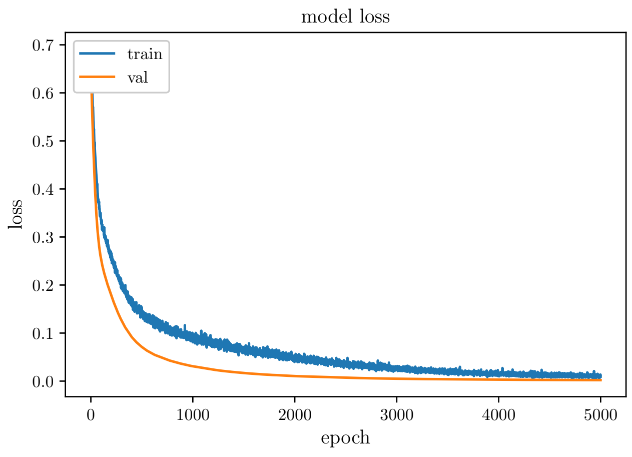

Fig. 9. SPC MLP

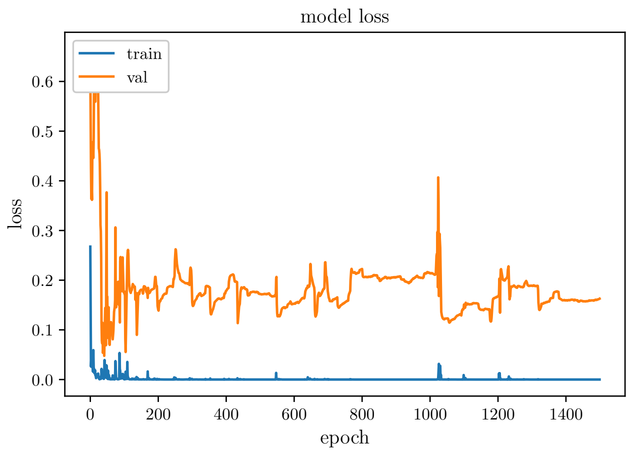

Fig. 10. SPC ResNet

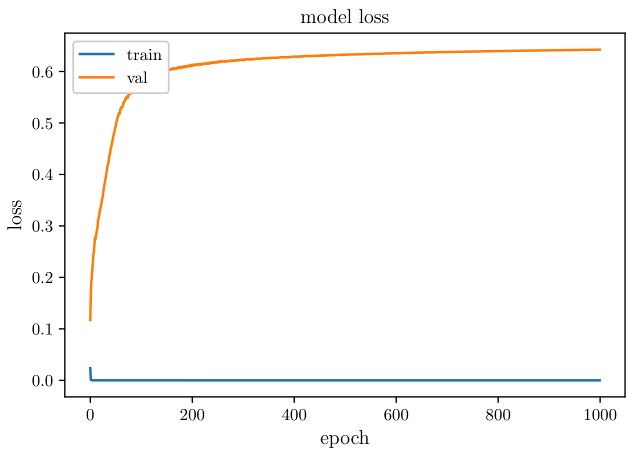

Fig. 11. SPC t-LeNet

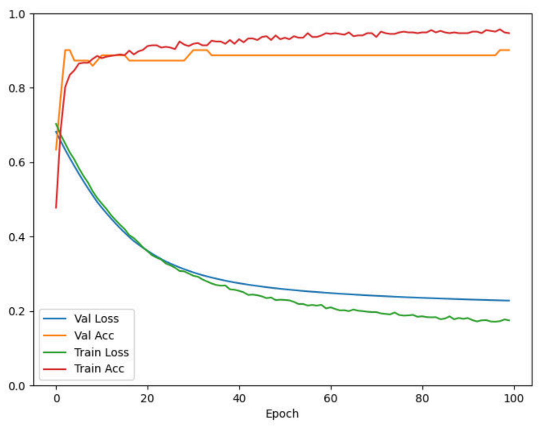

Fig. 12. SPC 2 LSTM + Dense

FCN, Inception and ResNet models' performance can be improved by using more complex data pre-processing techniques and increasing the data set size.

In Figure 12 a RNN implementation consists of 2 LSTM layers and 1 Dense layer was tested and resulted in 90%

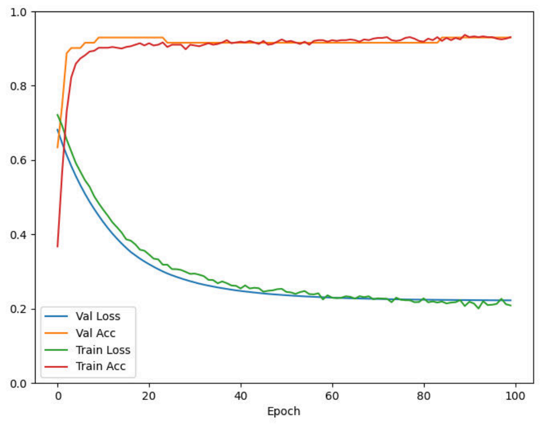

Fig. 13. SPC 2 GRU + Dense

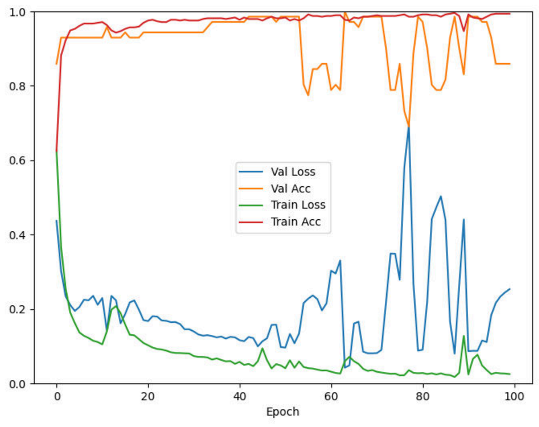

Fig. 14. SPC 2 RNN + Dense

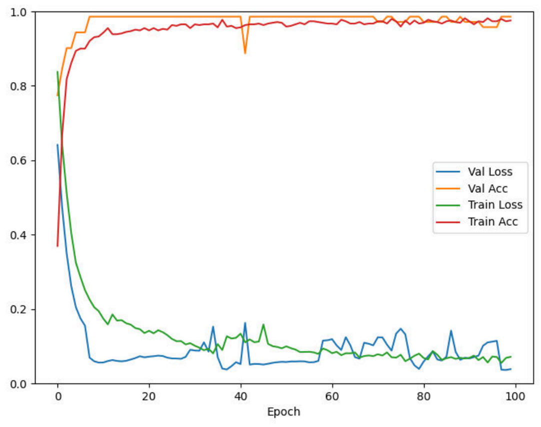

Fig. 15. SPC 1 RNN + Dense

validation accuracy.

TABLE I
PERFORMANCE OF SPCS BASED MODELS

Model	Val_Acc	Precision	Recall	F1
CNN	0.99	0.991	0.989	0.99
Encoder	0.985	0.981	0.974	0.977
FCN	0.9	0.907	0.909	0.908
Inception	0.985	0.985	0.986	0.985
MCDCNN	0.981	0.979	0.979	0.979
MCNN	0.589	0.294	0.5	0.37
MLP	1	1	1	1
ResNet	0.976	0.973	0.975	0.974
t-LeNet	0.987	0.984	0.986	0.985
TWIESN	0.985	0.988	0.982	0.985
2 LSTM	0.901	0.891	0.953	0.921
2 GRU	0.929	0.913	0.976	0.943
2 RNN	1	1	1	1
1 RNN	0.985	1	0.976	0.988

In Figure 13 a RNN implementation consists of 2 GRU layers and 1 Dense layer was tested and resulted in 93% validation accuracy.

In Figure 14 a RNN implementation consists of 2 simple RNN layers and 1 Dense layer was tested and resulted in 100% validation accuracy.

In Figure 15 a RNN implementation consists of 1 simple RNN layer and 1 Dense layer was tested and resulted in 98% validation accuracy.

The overall performance of all the models definitely will be improved after increasing the data set size.

In Table I are presented the SPC models performances.

B. *Hardware Performance Counters*

In this section we used a data from a different experiment [22]. In this experiment the data have been collected using a program called Savitor, which is used to extract AMD Hardware Performance Counters readings. They ran 962 malware and 962 benign software, HPCs were measured 32 times for each program. Each program was kept running for a minute and HPC readings were measured 32 times in equal intervals.

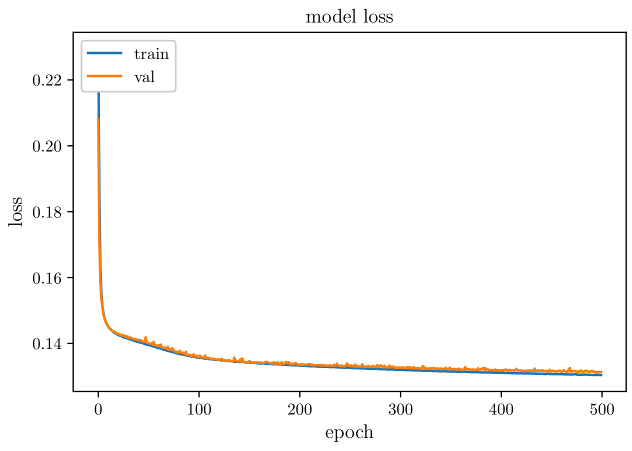

Fig. 16. HPC CNN

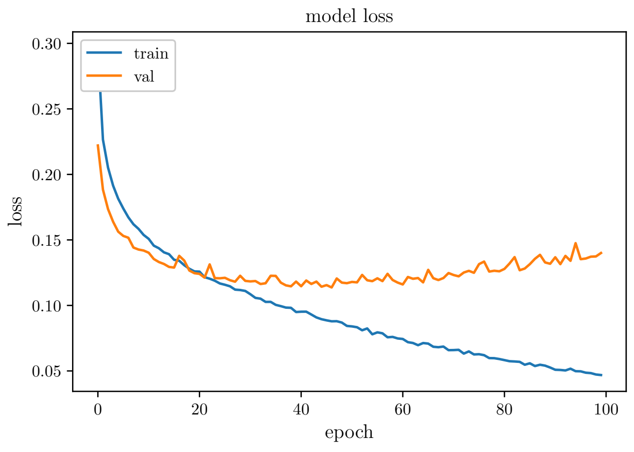

Fig. 17. HPC Encoder

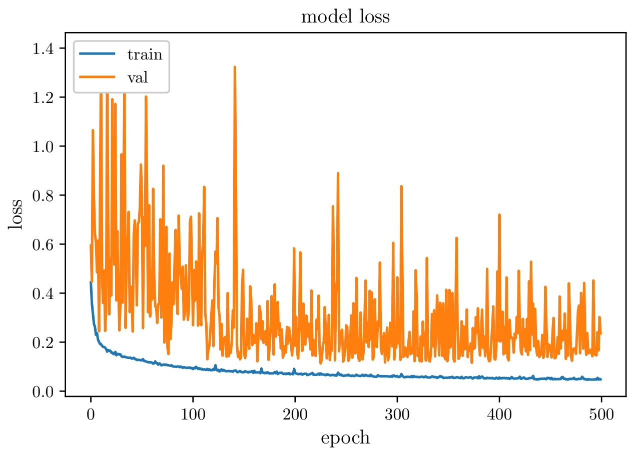

Fig. 18. HPC FCN

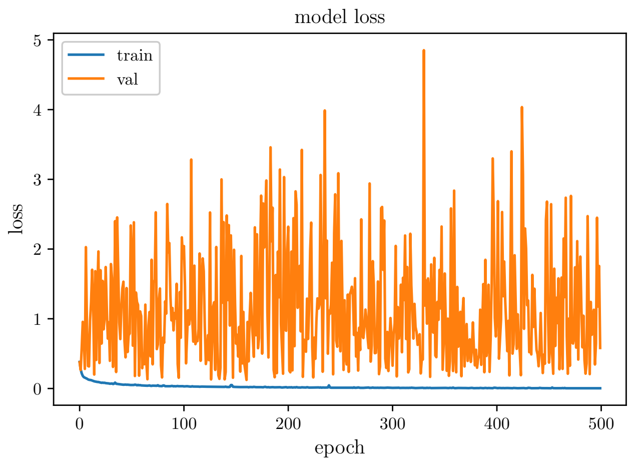

Fig. 19. HPC Inception

In Figures 16, 17, 18, 19, 20, 21 are presented the HPC models loss functions.

In Table II are presented the relevant HPC models performances.

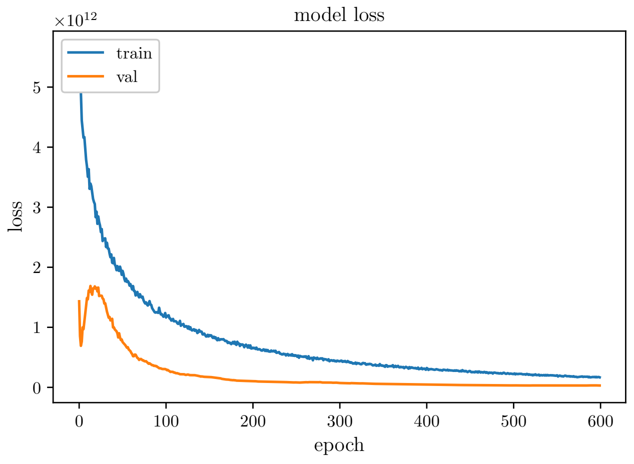

Fig. 20. HPC MLP

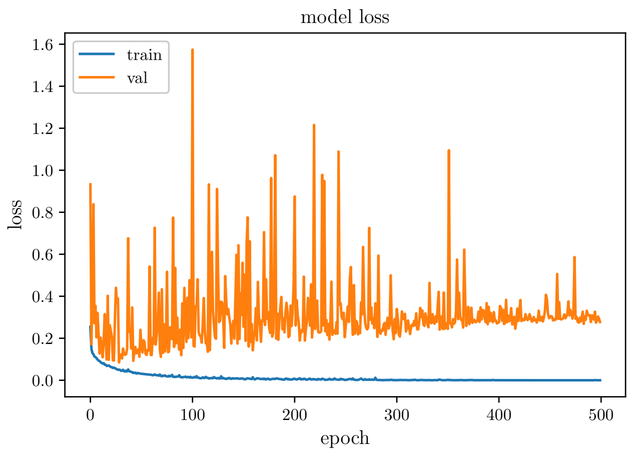

Fig. 21. HPC ResNet

V. Discussion

The results are affected by 2 major factors, data samples number and the similar processes in the operating system.

Apparently, our dataset consisting of 700 samples wasn't enough for some classifiers and led to overfitting.

The second problem is the interference of the malware downloader and the fake children processes that create the main malware process.

TABLE II
Performance of HPCs based models

Model	Val_Acc	Precision	Recall	F1
CNN	0.853	0.829	0.809	0.818
Encoder	0.959	0.957	0.0.959	0.958
MLP	0.672	0.687	0.687	0.687

These processes are very similar in terms of counters' data to a lot of legitimate MS Windows services' processes and some other processes like auto updaters.

We can see that this problem affected FCN, t-LeNet, ResNet, and FCN networks as they couldn't differentiate between these processes in the validation set.

This problem can be solved by using different approaches in the counters extraction process like identifying the processes that causes the problem and ignoring them so that we end up only with the effective and real processes.

Also increasing the size and variety of the data set will improve the overall performance of the model.

Despite these problems, a couple of models achieved promising results.

VI. Conclusions and Future Work

In this paper we presented an experimental setup where software and hardware performance counters time series were extracted from a safe and a malware infected virtual machine obtained from VirusTotal.

The performance counter time series were normalized and were used to train 10/14 SPCs/HPCs deep learning classification models.

For SPCs Encoder and MCDCNN models gave the best results, while the other models tended to overfit.

For HPCs CNN and MLP gave the best results, while Encoder gave best results till 50 epochs aftewards started to overfit the training data.

As future work we intend to test more malware and to increase the data set size to avoid overfitting problems.

A different malware categories will be tested to increase the variation of the data set and try to mimic a real-time anti-virus.

Acknowledgment

This paper was partially supported by a grant of the Romanian Ministry of Research, Innovation and Digitization, project number POC3981-Development of networks of R&D centers, coordinated at national level and connected to European and international networks and ensuring researchers' access to European and international scientific publications and databases, entitled "High Performance Cloud Platform at Politehnica University of Timisoara", SMIS 123466.

References

[1] Shekoofeh Azizi, Sharareh Bayat, Pingkun Yan, Amir Tahmasebi, Jin Tae Kwak, Sheng Xu, Baris Turkbey, Peter Choyke, Peter Pinto, Bradford Wood, et al. Deep recurrent neural networks for prostate cancer detection: Analysis of temporal enhanced ultrasound. *IEEE transactions on medical imaging*, 2018.

[2] Kanad Basu, Prashanth Krishnamurthy, Farshad Khorrami, and Ramesh Karri. A theoretical study of hardware performance counters-based malware detection. *IEEE Transactions on Information Forensics and Security*, 15:512–525, 2020.

[3] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification, 2016.

[4] Sai Manoj Pudukotai Dinakarrao, Sairaj Amberkar, Sahil Bhat, Abhijitt Dhavlle, Hossein Sayadi, Avesta Sasan, Houman Homayoun, and Setareh Rafatirad. Adversarial attack on microarchitectural events based malware detectors. In *In The 56th Annual Design Automation Conference 2019 (DAC '19)*, pages 1–6, Las Vegas, NV, USA, June 2019. ACM, New York, NY, USA.

[5] Amazon Inc. Amazon Web Services. https://aws.amazon.com, 2021.

[6] Oracle Inc. Virtual Box. https://www.oracle.com, 2021.

[7] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Jul 2019.

[8] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, Sep 2020.

[9] Sai Praveen Kadiyala, Akella Kartheek, and Tram Truong-Huu. Program behavior analysis and clustering using performance counters, 2021.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[11] Partha Pratim Kundu, Lux Anatharaman, and Tram Truong-Huu. An empirical evaluation of automated machine learning techniques for malware detection. In *In Proceedings of the 2021 ACM International Workshop on Security and Privacy Analytics (IWSPA'21)*, pages 1–7. ACM, New York, NY, USA, April 2021.

[12] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data Augmentation for Time Series Classification using Convolutional Neural Networks. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, Riva Del Garda, Italy, September 2016.

[13] Omar Mohamed and Ciprian-Bogdan Chirila. Towards malware detection based on performance counters using deep learning classification models. In *2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 000149–000154, 2022.

[14] Nazarii, Taras Lutsiv, Mykola Maksymyuk, Orest Beshley, Volodymyr Lavriv, Anatoliy Andrushchak, Liberios Sachenko, Juraj Vokorokos, and Gazda. Deep semisupervised learning-based network anomaly detection in heterogeneous information systems. *Computers, Materials & Continua*, 70(1):413–431, 2022.

[15] Nisarg Patel, Avesta Sasan, and Houman Homayoun. Analyzing hardware based malware detectors. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[16] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. Towards a universal neural network encoder for time series. *CoRR*, abs/1805.03908, 2018.

[17] Rashid Tahir, Sultan Durrani, Faizan Ahmed, Hammas Saeed, Fareed Zaffar, and Saqib Ilyas. The browsers strike back: Countering cryptojacking and parasitic miners on the web. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 703–711, April 2019.

[18] Virus Total Enterprise. Virus collection. https://www.virustotal.com, 2021.

[19] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585, 2017.

[20] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[21] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *WAIM*, 2014.

[22] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. Hardware performance counters can detect malware: Myth or fact? In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, page 457–468, New York, NY, USA, 2018. Association for Computing Machinery.