Static Analysis for Malware Classification Using Machine and Deep Learning

M.I.P. Salas*†, Paulo de Geus*
* *University of Campinas*, Campinas, SP, Brazil - Email: {mpalma, paulo}@lasca.ic.unicamp.br
† *Universidad Mayor de San Andrés*, La Paz, Bolivia

***Abstract*—Malware, or malicious software, is a general term to describe any program or code that can be harmful to systems. This hostile, intrusive, and intentionally harmful code makes use of a variety of techniques to protect and evade detection and removal through code obfuscation, polymorphism, metamorphism, encryption, encrypted communication, and more. Current state-of-the-art research focuses on the application of artificial intelligence techniques for the detection and classification of malware. In this context, this paper proposes a new malware classification through static analysis using seven machine learning algorithms (LightGBM, XGBoost, Logistic Regression, KNN, SVM, Naive Bayes, and Random Forest) and deep learning fine-tuning. These models make use of the SelectKBest technique within data engineering, allowing the selection of the 892 most relevant characteristics for the classification of 10868 malware in 9 families, reducing overfitting and training time. The results show that the application of Gradient Boosting algorithms such as LightGBM with hyperparameter optimization exceeds the reference results in competitions such as Kaggle, with a logarithmic loss 0.00118, an accuracy close to 100%, and prediction times less than 2.3ms. Fast enough to be applied to systems in real time to classify malware.**

***Index Terms*—malware classification, static analysis, SelectKBest, lightgbm, machine learning, deep learning**

I. Introduction

In the era of the Internet of Things (IoT), where every device is a functional computer susceptible to malware infection, there is a need for new detection techniques. Current antivirus solutions are highly effective and absolutely essential for detecting and eliminating malware. However, they have a weakness: they need to wait for samples of a particular malware to be obtained in order to be analyzed, create its signature, establish a cleaning procedure, and thus be able to eliminate it from production systems [1], [2].

To detect and classify each new piece of malware, antivirus companies apply a reactive approach, which involves collecting samples through honeypots or antivirus software running on client machines [3]. Malware samples are stored and executed in automated analysis systems, such as Cuckoo [4], providing information on their behavior, flaws, and errors that abuse applications and operating systems. With this information, antivirus software can be extended to identify the new malware and classify it.

The increased connectivity between applications and devices has brought about a greater spread of malware, e.g., Emotets [5]. As a result, thousands of devices became infected within hours, reducing the effectiveness of antivirus software to detect new threats and requiring constant updates to their signatures.

New methods and techniques of obfuscation, polymorphism, and compression are used by developers to make malware detection more difficult and to disable security software (such as IDS, antivirus, and firewalls, among others) [6]. The high increase in malware, not only for Windows OS and Android OS, allows us to see that this security problem needs further research.

Machine learning and deep learning allow to analyze the pattern of behavior and activities of an application to be classified within a family of malware, particularly using static and dynamic analysis. Thus, by verifying that a program is performing potentially harmful actions, it is possible to classify it, analyze it, and better understand the security threats and risks. Consequently, it will help improve security solutions and strategies to prevent and mitigate future attacks.

In this research, we propose a new methodology for malware classification of portable executable (PE) files through static analysis using seven machine learning algorithms (LightGBM, XGBoost, Logistic Regression, KNN, SVM, Naive Bayes, and Random Forest) and deep learning with fine-tuning for the classification of 10,868 malware samples into 9 families. To reduce overfitting, we use the SelectKBest technique within data engineering, reducing the number of features from 84,611 to just 892 more relevant features, controlling the training time.

The results show that the application of gradient boosting algorithms, such as LightGBM or XGBoost, with hyperparameter optimization improves reference results in competitions such as the Microsoft Malware Classification Challenge [7] on Kaggle, reducing the best score from 0.00283 to 0.00118, with accuracy close to 1 and execution times of less than 2.3 ms. It is fast enough to be applied to gadgets in real-time to classify malware.

The rest of the document is divided as follows: Section II describes the related work on malware classification using machine learning approaches. In Section III, we analyze the PE files and describe the evolution of malware. Section IV describes the software analysis processes, delving into static analysis. The malware classification methodology is developed in Section V, concluding with the research results. The main contributions and future work are described in Section VI.

II. Related work

According to recent studies [5], [8], there is an alarming increase in malware attacks. The increase in attacks is due to a combination of factors, including the growth of internet-connected devices, the increase of remote work as a work modality after the COVID-19 pandemic started, and the use of more sophisticated techniques by attackers.

Machine learning can be an attractive tool for the detection and classification of malware. Thus, Hyrum and Phil [9] present EMBER, a benchmark dataset for training machine learning models to detect malicious Windows portable executable (PE) files statically. The database of over one million benign and malignant samples was tested with the LightGBM algorithm with default parameters (100 trees, 31 leaves per tree), resulting in fewer than 10K tunable parameters and a detection rate of about 93% without hyper-parameter optimization.

In [10], Barker et al. present MalConv, a convolutional neural network architecture that works by processing entire executable files in raw byte format to detect malware. This technique focuses on analyzing malicious binary files to detect the presence of hidden patterns in their content through the use of byte n-grams and PE-Header as important information. Being one of the first deep network architectures for malware detection, it achieved 92.5% accuracy in malware identification.

Gibert et al., in [11], use a set of metamorphic malware to test the accuracy of known CNN architectures in malware detection and find that these models are susceptible to metamorphic attacks. Thus, the authors propose a shallow CNN architecture to counteract this code obfuscation technique, obtaining 97.48% under the mentioned architecture and 97.65% with data aggregation.

In [12], the authors introduce an innovative real-time malware detection approach based on network data stream analysis using a convolutional neural network architecture. Furthermore, they propose a learning transfer approach through pre-trained networks to improve the accuracy of malware detection. Both the detection and classification of malware are evaluated in machine learning and deep learning architectures using different databases, reaching an accuracy of 98.4

Han et al. [13] present a systematic approach to detecting malware using three-level characteristics (basic or static, low-level behavior, and high-level behavior) of malicious and benign files rather than relying solely on virus signatures or file structure characteristics. After extracting relevant information from the malware through Cuckoo SandBox, the authors use four machine learning models, concluding that Random Forest has the best accuracy (97.21%). This research shows that the robustness of the malware analysis model lies in the extraction and categorization of malicious code features.

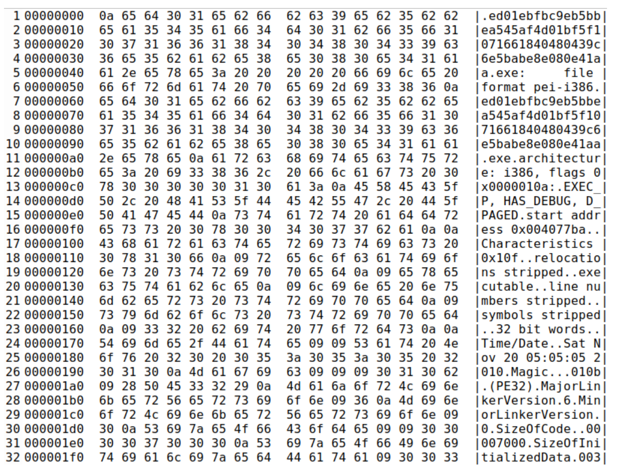
The authors in [14] discusses the traditional approaches used for malware detection, including virus signatures, heuristics, and dynamic analysis. More advanced approaches, such as static and dynamic analysis based on machine learning and deep learning, are also described. It is concluded that none of the traditional and more advanced approaches can detect and classify all types of malware, especially zero-days. New researchers will be guided to use combined approaches to create new architectures.

III. Background

Malware is malicious software that can cause serious damage to computer systems and user privacy. In this section, we justify the malware classification research.

A. PE File Format

The PE (Portable Executable) file format is an executable file standard used in 32 and 64 bit in Windows operating systems. The term portable refers to the format's versatility in numerous operating system software architecture environments. On Windows NT-based, the file types EXE, DLL, SYS (device drivers), and other, used the PE format [15].



```
00000000  0a 65 64 30 31 65 62 66  62 63 39 65 62 35 62 62  |.ed01ebfbc9eb5bb|
00000010  65 61 35 34 35 61 66 34  64 30 31 62 66 35 66 31  |ea545af4d01bf5f1|
00000020  30 37 31 36 36 31 38 34  30 34 38 30 34 33 39 63  |071661840480439c|
00000030  36 65 35 62 61 62 65 38  65 30 38 30 65 34 31 61  |6e5babe8e080e41a|
00000040  61 2e 65 78 65 3a 20 20  20 20 20 66 69 6c 65 20  |a.exe:     file |
00000050  66 6f 72 6d 61 74 20 70  65 69 2d 69 33 38 36 0a  |format pei-i386.|
00000060  65 64 30 31 65 62 66 62  63 39 65 62 35 62 62 65  |ed01ebfbc9eb5bbe|
00000070  61 35 34 35 61 66 34 64  30 31 62 66 35 66 31 30  |a545af4d01bf5f10|
00000080  37 31 36 36 31 38 34 30  34 38 30 34 33 39 63 36  |71661840480439c6|
00000090  65 35 62 61 62 65 38 65  30 38 30 65 34 31 61 61  |e5babe8e080e41aa|
000000a0  2e 65 78 65 0a 61 72 63  68 69 74 65 63 74 75 72  |.exe.architectur|
000000b0  65 3a 20 69 33 38 36 2c  20 66 6c 61 67 73 20 30  |e: i386, flags 0|
000000c0  78 30 30 30 30 30 31 30  61 3a 0a 45 58 45 43 5f  |x0000010a:.EXEC_|
000000d0  50 2c 20 48 41 53 5f 44  45 42 55 47 2c 20 44 5f  |P, HAS_DEBUG, D_|
000000e0  50 41 47 45 44 0a 73 74  61 72 74 20 61 64 64 72  |PAGED.start addr|
000000f0  65 73 73 20 30 78 30 30  34 30 37 37 62 61 0a 0a  |ess 0x004077ba..|
00000100  43 68 61 72 61 63 74 65  72 69 73 74 69 63 73 20  |Characteristics |
00000110  30 78 31 30 66 0a 09 72  65 6c 6f 63 61 74 69 6f  |0x10f..relocatio|
00000120  6e 73 20 73 74 72 69 70  70 65 64 0a 09 65 78 65  |ns stripped..exe|
00000130  63 75 74 61 62 6c 65 0a  09 6c 69 6e 65 20 6e 75  |cutable..line nu|
00000140  6d 62 65 72 73 20 73 74  72 69 70 70 65 64 0a 09  |mbers stripped..|
00000150  73 79 6d 62 6f 6c 73 20  73 74 72 69 70 70 65 64  |symbols stripped|
00000160  0a 09 33 32 20 62 69 74  20 77 6f 72 64 73 0a 0a  |..32 bit words..|
00000170  54 69 6d 65 2f 44 61 74  65 09 09 53 61 74 20 4e  |Time/Date..Sat N|
00000180  6f 76 20 32 30 20 30 35  3a 30 35 3a 30 35 20 32  |ov 20 05:05:05 2|
00000190  30 31 30 0a 4d 61 67 69  63 09 09 09 30 31 30 62  |010.Magic...010b|
000001a0  09 28 50 45 33 32 29 0a  4d 61 6a 6f 72 4c 69 6e  |.(PE32).MajorLin|
000001b0  6b 65 72 56 65 72 73 69  6f 6e 09 36 0a 4d 69 6e  |kerVersion.6.Min|
000001c0  6f 72 4c 69 6e 6b 65 72  56 65 72 73 69 6f 6e 09  |orLinkerVersion.|
000001d0  30 0a 53 69 7a 65 4f 66  43 6f 64 65 09 09 30 30  |0.SizeOfCode..00|
000001e0  30 30 37 30 30 30 0a 53  69 7a 65 4f 66 49 6e 69  |007000.SizeOfIni|
000001f0  74 69 61 6c 69 7a 65 64  44 61 74 61 09 30 30 33  |tializedData.003|
```

Fig. 1: The 32-bit PE file structure of ransomware WannaCry.

The PE format is a data structure that encapsulates the information needed by the Windows loader. This includes the dynamic linker to allocate the file in memory and create the references to the libraries, the import and export of API tables, resource data management, and thread local storage data (TLS data).

The PE file consists of a header and one or more segments, as shown in Fig. 1. The header contains important information about the file structure, such as the input address, the location of the segments, and the amount of resources and data it contains. The segments contain the executable code and the data necessary for the application to run correctly. These segments can include the application code, data tables, resources, and other components.

The executable code inside the PE file consists of opcodes that tell the CPU what action to perform, such as moving data between registers, performing arithmetic-logical operations, jumping to a specific memory address, and so on. These opcodes are inside the segments that contain the executable code. Additionally, executable code within the PE file can make use of registers, which are storage areas on the CPU that are used to store temporary data during the execution of an application.

PE files can also make use of function calls and APIs to access functions and resources in the OS and other software libraries. These calls can be made by executable code within a PE file to perform specific tasks required for the application to run, such as reading and writing files, interacting with the operating system and other programs, and performing network operations.

More information about the PE file format can be found in [16] and [15].

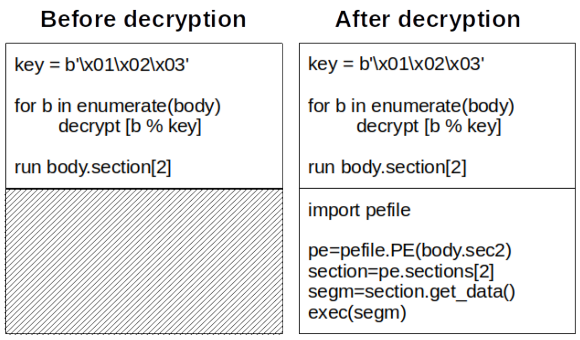
B. Types of Malware

Malware has evolved from simple viruses that spread across disks and networks to sophisticated cyberattacks that can steal personal information, extort money from users, and turn devices into zombies for third-party attacks. Here are some of the most common types [6]:

- A **virus** infects computers and files by replicating itself and spreading when that program is run, causing system performance degradation and denial of service.
- **Worms** are a self-replicating type of malware that can spread across networks, consuming network and computer resources, which can lead to system performance degradation.
- A **Trojan horse** disguises itself as legitimate software but actually contains malicious code. It can be used to steal data, spy on a device, or launch attacks on other systems.
- **Rootkits** allow hackers to gain privileged access to a system and hide their activities from detection.
- **Spyware** keeps watch on a user's activities and secretly collects information from a computer or device, often including personal data. It then sends the information back to the hacker.
- **Keyloggers** are a type of spyware used to record keystrokes and steal sensitive information like passwords and credit card numbers.
- **Adware** displays unwanted ads on a computer or mobile device, slowing down the system and being difficult to remove.
- A **backdoor** is designed to bypass a system's security mechanisms and install itself on a computer, allowing the attacker to access it.
- A **botnet** is a network of infected computers that can be controlled remotely by an attacker using a Command and Control (C&C) server. This malicious network uses a software called a bot that allows an attacker to control infected computer devices. Botnets can be used to carry out Distributed Denial of Service (DDoS) attacks, send spam messages, and steal information over the Tor network.
- **Ransomware** encrypts the victim's files and demands a ransom payment in exchange for the decryption key. There is no guarantee that the encrypted information will be returned.

C. Advanced Malware Protection Techniques

Malware techniques use a variety of methods to hide themselves from malware detectors, including [6]:



```
key = b'\x01\x02\x03'

for b in enumerate(body)
        decrypt [b % key]

run body.section[2]
```

```
key = b'\x01\x02\x03'

for b in enumerate(body)
        decrypt [b % key]

run body.section[2]

import pefile

pe=pefile.PE(body.sec2)
section=pe.sections[2]
segm=section.get_data()
exec(segm)
```

Fig. 2: Encrypting the code and decrypting it at runtime to avoid detection.

- **Polymorphism:** changing the code structure of the malware to evade signature-based detection.
- **Metamorphism:** changing the code structure and behavior of the malware at runtime to avoid static and dynamic analysis.
- **Code obfuscation:** modifying the code to make it difficult to read and analyze by human or automated tools.
- **Encryption:** encrypting the malware code and decrypting it at runtime to avoid detection, as shown in Fig. 2.
- **Rootkit:** modifying the operating system to hide the presence of the malware and its activities.
- **Virtualization:** running the malware in a virtual machine environment to avoid detection and analysis by malware detectors.
- **Anti-debugging:** detecting and avoiding analysis attempts by debugger tools to prevent detection.

D. Malware Detection Techniques

There are several techniques that can be used for malware detection. Combining these techniques can lead to more effective malware detection, classification, and prevention. Some of the techniques include:

- **Signature-based detection:** This technique involves comparing the digital signature or hash of a file to a known database of signatures of known malware. If a match is found, the file is flagged as malicious [2].
- **Heuristic-based detection:** This technique uses a set of rules or algorithms to analyze the behavior of a file or program and determine if it is likely to be malicious [2].
- **Behavior-based detection:** This technique involves monitoring the behavior of a file or program in real-time to detect any suspicious activity or deviations from normal behavior [8].
- **Machine learning-based detection:** This technique uses machine learning algorithms to analyze large datasets of known malware to detect new, classified and emerging threats [9], [11], [19], [19].

TABLE I: Advanced techniques to Protect Malware [17], [18].

Anti-Analysis techniques	Static Analysis Method				
	Code and Data Flow	Signature	API Calls and Function	Decompilation	Obfuscation
Polymorphism	✓	✓	-	-	-
Metamorphism	✓	✓	-	-	-
Code obfuscation	✓	✓	✓	✓	✓
Packing	-	✓	-	✓	✓
Anti-debugging	✓	✓	✓	✓	✓
Rootkit	✓	-	-	-	-
Virtualization	✓	-	-	✓	-
Encryption	-	-	✓	-	✓
Dynamic code generation	-	-	-	-	✓

- **Sandboxing:** This technique involves running a file or program in a virtual environment to observe its behavior and detect any malicious activity [4].
- **YARA rules:** YARA is a tool used for pattern matching of malware, enabling users to write custom rules for detecting specific malware [4].

IV. Malware Analysis

The processes used to study, identify, and understand the behavior and characteristics of malicious code are known as malware analysis. It involves studying malicious files in order to better understand various aspects of the malware, such as behavior, evolution over time, and selected targets [17]. Additionally, by using these processes, malware analysts can gain a better understanding of how malware works, its capabilities, and its potential impact and can help develop effective detection, classification, and mitigation strategies [17]. The following are some commonly used malware analysis processes described in Fig. 3:

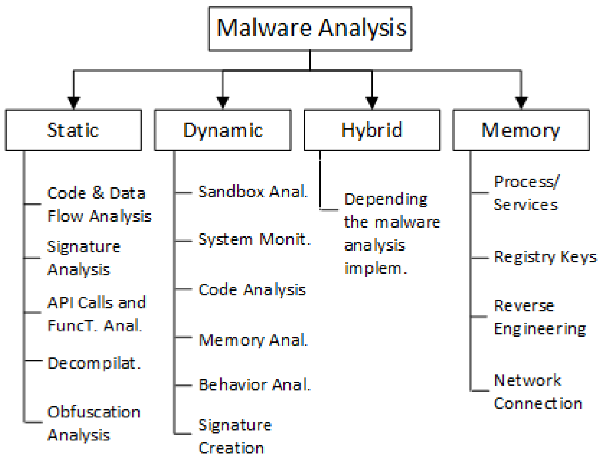


Fig. 3: Malware analysis processes and features.

A. *Static analysis*

This process involves examining the binary code and structure of a malware sample without executing it. The process involves analyzing the code structure, syntax, and other static properties to identify potential security vulnerabilities, malicious behavior, or other characteristics that can help in determining whether the code is malicious or not.

This process is effective in detecting well-known malware strains, especially when it is combined with other methods, e.g., decompilation analysis and API calls and function analysis, even when malware developers make use of advanced techniques to protect malware, as seen in Table I.

Static analysis can be performed manually or with the help of automated tools, as shown in Table II. It is often used in conjunction with dynamic analysis techniques to provide a comprehensive view of the malware's behavior. Commonly employed static analysis methods include:

- **Code and Data Flow Analysis:** It involves analyzing the code structure and how it interacts with data to understand the behavior of the malware. This technique helps to identify the malware's functionality, such as whether it steals data or creates a backdoor. The analyst can use a CFG (Control Flow Graph) to capture the behavior of a PE file and extract the program structure. The researcher can make use of Ghidra[1] and Radare2[2], among other tools, to apply this method.
- **Signature Analysis:** It involves the use of predefined signatures or patterns to identify known malware. It works by comparing the code or binary file of a suspected malware sample with a database of known malware signatures. To create a malware signature, the researches use tools as shown in Table II. This technique is effective in detecting well-known malware strains, but it is not useful in detecting new or unknown malware. The researcher can make use of Yara Rules[3], VirusTotal[4] and Snort[5], among other tools, to apply this method.
- **API Calls and Function Analysis:** It involves analyzing the API calls (short for Application Programming Interface) and function used by the malware to gain insights into its behavior. This technique helps in identifying the malware's capabilities, such as whether it has rootkit functionality or creates new processes. The researcher can make use of OllyDbg[6] and x64dbg[7], among other tools, to apply this method.

[1]https://ghidra-sre.org/
[2]https://rada.re/n/
[3]https://github.com/Yara-Rules/rules
[4]https://www.virustotal.com/gui/
[5]https://www.snort.org/
[6]https://www.ollydbg.de/
[7]https://x64dbg.com/

TABLE II: Tools for Malware Static Analysis [17], [18].

Tools for static analysis	**Static Analysis Method**				
	Code and Data Flow	**Signature**	**API Calls and Function**	**Decompilation**	**Obfuscation**
Antivirus software	-	✓	-	-	-
YARA	-	✓	-	-	-
VirusTotal	-	✓	-	-	-
Snort	-	✓	-	-	-
IDA Pro	✓	✓	✓	✓	✓
PEiD	-	✓	-	-	-
Sysinternals	-	✓	-	-	-
Ghidra	✓	-	-	✓	-
OllyDbg	-	-	✓	-	✓
x64dbg	-	-	✓	-	-
Radare2	✓	-	✓	-	✓
Opcode	✓	✓	-	✓	✓
N-grams	✓	-	✓	✓	✓

- **Decompilation:** It involves converting the malware code into a high-level programming language to understand its behavior better. This technique helps in identifying the malware's functionality and the vulnerabilities it exploits. The researcher can make use of IDA Pro and Opcodes[8], among other tools, to apply this method.
- **Obfuscation Analysis:** It involves identifying and analyzing the obfuscated code to reveal the malware's true behavior. This technique helps to identify the malware's capabilities, such as whether it tries to hide its presence or uses anti-debugging techniques. The researcher can make use of IDA Pro and OllyDbg, among other tools, to apply this method.

Static analysis can improve the malware classification process by providing a quick and efficient way to analyze large numbers of files. By using automated tools to scan files for known malware signatures or suspicious behavior, analysts can quickly triage files and focus their efforts on the most promising cases.

Static analysis can also be used to extract features from malware samples that can be used to train machine learning models for classification. By analyzing the code or binary of a malware sample, features such as function calls, API calls, and string literals can be extracted and used as input to a machine learning model. These models can then be used to classify new malware samples based on their similarity to known malware families or behaviors.

Some techniques used in static analysis are described below.

1) Opcodes: are an integral part of a machine code instruction, which specifies the operation that the CPU needs to perform. A complete machine language instruction consists of an opcode along with one or more operands, such as ''mov eax 7'', ''add eax ecx'', and ''sub ebx 1''. Opcodes can be utilized as a feature in malware detection by analyzing the frequency of opcodes or determining the similarity between opcode sequences [17].

In opcode analysis, the malware analyst examines the opcode sequences to identify patterns and understand the function of the code. For example, a sequence of opcodes may indicate that the code is opening a network connection, downloading additional code, or modifying system files. This information can be used to identify the malware's capabilities and potential impact on a system.

Opcode analysis can be performed manually, by examining the assembly language code line by line, or with the help of automated tools, such as disassemblers or decompilers. These tools can identify and extract the opcodes from the code, making it easier for analysts to identify patterns and understand the behavior of the malware.

2) N-grams: analysis is a technique used in natural language processing and text mining to analyze and understand the structure of text data. It involves breaking down a text into contiguous sequences of n items, such as words or characters, and then counting the frequency of occurrence of each sequence [18].

For example, in a sequence of words like ''This program cannot be run in DOS mode'', the 3-grams would be: ''This program cannot'', ''program cannot be'', ''cannot be run'', ''be run in'', and ''run in DOS'', ''in DOS mode''.

In malware analysis, n-grams analysis is used to identify patterns and similarities between different samples of malware. By analyzing the frequency of occurrence of n-grams in different samples, researchers can identify common sequences of code or behavior that may indicate the presence of malware.

B. Dynamic analysis

Dynamic analysis involves analyzing the behavior of malware as it runs on a system or in a controlled environment. This approach involves executing the malware in a sandbox, bare metal[9] environment, or virtual environment, which allows the analyst to observe its behavior without risking infection or damage to the host system [6].

During dynamic analysis, the analyst monitors the malware's actions, such as file system modifications, network communication, process creation, and registry changes. They may use tools such as debuggers, disassemblers, and network monitors to capture and analyze this behavior. Also, this process can help analysts understand the purpose and

[8]http://ref.x86asm.net/coder32.html

[9]A ''bare metal'' environment is a physical machine on which the operating system is installed directly on the hardware without any hypervisor in between.

functionality of the malware, identify its command and control infrastructure, and determine the extent of its impact on the system.

This malware analysis use uses several techniques and processes to analyze malware behavior during execution. Some of these techniques and processes are:

- **Sandbox Analysis:** involves running the malware in a controlled environment, such as a virtual machine, bare metal or a sandbox, to observe its behavior. This approach helps to protect the host system from infection and damage caused by the malware.
- **System Monitoring:** The malware's behavior is monitored during this analysis using tools such as debuggers, disassemblers, and network monitors. These tools help to capture the malware's interactions with the system and network.
- **Code Analysis:** involves analyzing the malware's code as it executes. This can help in identifying specific functions, system calls, and other behaviors that are used by the malware.
- **Memory Analysis:** involves examining the malware's interactions with the system's memory. This can help in identifying memory-based attacks, such as buffer overflows or injection attacks.
- **Behavioral Analysis:** involves examining the malware's behavior to determine its purpose, capabilities, and intended targets. This approach can help in identifying the type of malware, such as a virus, worm, or Trojan.
- **Signature Creation:** can be used to create signatures or rules that can be used to detect and prevent similar malware attacks. These signatures can be used to enhance the efficacy of existing security tools and to develop new security mechanisms.

Dynamic malware analysis uses a combination of these techniques and processes to gain an in-depth understanding of the malware's behavior during execution.

Malware developers use a range of techniques to protect their malicious code, including anti-debugging techniques, code obfuscation, and rootkit techniques, among others.

C. Hybrid analysis

Hybrid analysis in malware analysis refers to the use of a combination of two or more different analysis techniques, e.g., static and dynamic analysis, to examine and understand a malware sample. This approach is used to overcome the limitations of individual analysis techniques and gain a more complete and accurate understanding of the behavior and capabilities of the malware [13], [17].

This process can provide a more complete picture of the malware's features, including its code, execution flow, network communications, and system-level interactions to detect, classify, and delete malware, providing a more comprehensive and accurate understanding of the malicious sample's behavior and capabilities.

Typically, hybrid malware analysis includes one or more of the following processes: static analysis, dynamic analysis, sandboxing analysis, memory analysis, and network analysis.

In the case of hybrid analysis, malware developers will make use of protection techniques depending on the analysis techniques that can be implemented by malware analyzers.

D. Memory analysis

This process provides a comprehensive examination of malware hooks and code outside the function's normal scope. It uses memory images to analyze information about running programs, the operating system, and the general state of the computer. Memory forensics investigations pass through two steps: memory acquisition and memory analysis. In memory acquisition, the memory of the target machine is dumped to obtain a memory image using tools such as Memoryze, FastDump, and DumpIt. The memory analysis step is to analyze the memory image, looking for malicious activities, using tools like Volatility and Rekall [17], [6].

Memory analysis can be conducted using a variety of tools and techniques, including specialized memory analysis software, reverse engineering tools, and manual analysis of memory dumps. It requires a deep understanding of operating system internals, malware behavior, and memory management principles.

Memory analysis employs various techniques to protect itself from memory analysis and evade detection. Some of the common techniques used by malware developers to protect themselv es from memory analysis include anti-debugging and anti-virtualization techniques, anti-memory forensics techniques, rootkit techniques, and obfuscation techniques.

V. Methodology

A. Experiment settings

The runtime environment of the experiment includes (1) ASUS nv580vd with an Intel® Core™ i7 7700HQ 2,8GHz processor, 16 GB SDRAM, NVIDIA GeForce GTX 1050, 4GB GDDR5 VRAM, Ubuntu 20.04 LTS (64bit). (2) i440fx-xenial. Intel Core i7 9xx (Nehalem Core i7, IBRS update), 8GB, Ubuntu 22.04 LTS.

B. Dataset Description

For the experiment we used the well-known Microsoft Malware Classification Challenge database. [7]. The experiment dataset is almost 200GB, consisting of a set of 21736 known malware files representing a mix of 9 different families, as shown in Table III. Each malicious file has a 20 character hash value uniquely identify, and a class label (1 to 9) representing one of the nine family names.

Half of the malicious fileset (10868) is made up of raw data with a hexadecimal representation of the file's binary content, without the header (to ensure sterility).

The rest of the malware samples (10868) are assembly code files (ASM) obtained by reverse engineering with the IDA disassembler tool[10]. These files contain metadata with

[10]https://hex-rays.com/ida-pro/

TABLE III: Malware Samples in the Dataset

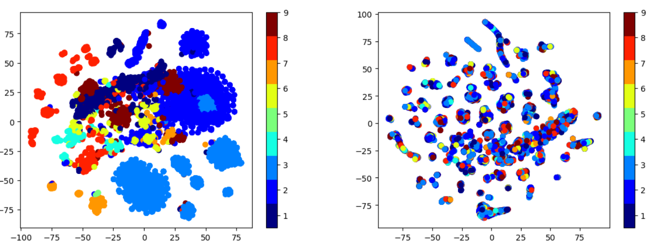
Family Name	# Train Samples	Type
Ramnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos_ver1	398	Backdoor
Obfuscator.ACY	1228	Any kind of obfuscated malware
Gatak	1013	Backdoor

addresses, segments, opcodes, registers, function calls, APIs, etc.

The data set can be downloaded from the competition website[11].

C. *Malware Feature Extraction*

First, a multivariate analysis was applied to analyze and model the set of files with bytes and asm extensions in order to find interdependent variables that could show the identification of the nine malware families or classes in isolation. It will not be possible to identify complex relationships or patterns in both files, as can be seen in Fig. 4.



(a) Multivariate Analysis on bytes files.

(b) Multivariate Analysis on ASM files.

Fig. 4: Application of multivariate analysis in the bytes and asm files to identify interdependent variables between the nine malware families.

The next step was the application of data mining for feature extraction. This data analysis process has the objective of identifying patterns and features in a data set that, before being processed, were unknown [14].

Therefore, it was decided to use N-grams for feature extraction. This natural language processing (NLP) and data mining technique was used to analyze and understand the bytes and ASM files (more information in IV-A2). This technique was selected because it can be used in combination with other techniques, such as machine learning and statistical techniques, in order to identify patterns and develop more sophisticated models for malware detection and classification [20].

It was decided to apply feature extraction of both file types (bytes and ASM):

- Byte files' size: size in Mb of the 10868 byte files.
- Byte files' unigram: Individual count of hexadecimal values in byte files.
- Byte files' bigram: count of two consecutive hexadecimal values in bytes files.
- ASM files' size: size in Mb of the 10868 asm files.
- key words' individual count of consecutive x86 instructions consisting of registers, memory, and and Opcodes in ASM files.
- ASM Opcodes' bi-gram: count of two consecutive Opcodes in ASM files.
- ASM Opcodes' tri-gram: count of three consecutive Opcodes in ASM files.

An analysis of the x86 assembly was carried out within the ASM files, obtaining a total of 51 X86 instructions composed of a set of 26 Opcodes, 13 prefixes, 9 registers, and 3 keywords, as shown in Table IV.

TABLE IV: X86 instructions consisting of registers, memory, and, Opcodes in ASM files.

#	x86 Instr.	#	x86 Instr.	#	x86 Instr.
1	HEADER:	18	pop	35	jnb
2	.text:	19	xor	36	jz
3	.Pav:	20	retn	37	rtn
4	.idata:	21	nop	38	lea
5	.data:	22	sub	39	movzx
6	.bss:	23	inc	40	.dll
7	.rdata:	24	dec	41	std::
8	.edata:	25	add	42	:dword
9	.rsrc:	26	imul	43	edx
10	.tls:	27	xchg	44	esi
11	.reloc:	28	or	45	eax
12	.BSS:	29	shr	46	ebx
13	.CODE	30	cmp	47	ecx
14	jmp	31	call	48	edi
15	mov	32	shl	49	ebp
16	retf	33	ror	50	ebp
17	push	34	rol	51	eip

Malware classification uses Opcode analysis techniques to examine the assembly language code of a program and identify its behavior by analyzing the instruction sequences, or operation codes, that are used in the code. Operation codes[12] are the building blocks of machine language and represent specific operations performed by a processor, such as moving data, adding or subtracting values, and branching to other sections of code. They also allow malware to be identified, classified, and removed in conjunction with other techniques, such as machine learning.

Prefixes[13] help assemblers and compilers organize and map the different parts of the program in memory and ensure that they run correctly during program execution on the x86 architecture. Each of these segments fulfills a specific function and contains instructions and data related to its purpose.

Registers[14] act as temporary storage for data and instructions during the execution of a program. Its direct and fast

[11]https://www.kaggle.com/competitions/malware-classification/

[12]wiki.osdev.org/X86-64_Instruction_Encoding

[13]https://wiki.osdev.org/X86-64_Instruction_Encoding

[14]https://www.cs.virginia.edu/ evans/cs216/guides/x86.html

access allows efficient performance in data processing and operations at the lowest level of the processor architecture.

TABLE V: Features used to classify malware families.

Feature name	Total	Best tf. slct.	%
Byte files' size	1	1	100.0%
Byte files' unigram	256	256	100.0%
Byte files' bi-gram	65536	330	0.5%
ASM files' size	1	1	100.0%
ASM instructions' unigram	51	51	100.0%
ASM Opcodes' bi-gram	676	77	11.4%
ASM Opcodes' tri-gram	17576	176	10.0%
TOTAL	84098	892	1.1%

The next step was to use the SelectKBest[15] function with the Chi-squared statistical test to improve the accuracy of the model and reduce training time. This feature selection function in machine learning was used to select the best "k" features from the set of features extracted by data mining and n-grams from Table V.

By reducing the dimensionality of the data, it was possible to reduce overfitting, reducing from 84098 features to only 892, or 1.1%, of the features obtained by data mining, guaranteeing that the selected data set is generalizable or representative for the application of machine learning models and malware classification.

D. Evaluation Metrics

In order to evaluate the performance with static analysis and data mining, we use logloss. This evaluation measure, commonly used for classification problems, allows us to measure the discrepancy between the probabilities predicted by the models used and the actual classes or families of malware. For each instance in the data set, the log loss measured the negative log probability that the model predicted the correct family of malware. The smaller the log loss value, the better the model will be in terms of classification accuracy.

The log loss is defined as:

$$log\_loss = -(1/N) * \sum_{i=1}^{N} [y_i * \log(y_{pred,i}) + (1 - y_i) * \log(1 - y_{pred,i})] \tag{1}$$

Where:

- N is the total number of instances in the data set.
- y_i is the actual class variable (0 or 1) for the i instance.
- $y_{pred,i}$ is the probability predicted by the model that instance i belongs to class 1.

Equation 1 shows the formula for log loss. This evaluation measure can range from 0 to infinity, where a value of 0 indicates a perfect classification and an infinity value indicates a complete misclassification. In practice, log loss values are generally positive, and the closer the values are to zero, the better the model performs.

[15]https://scikit-learn.org

E. Malware Classification

In recent years, there has been a rapid increase in the number of academic studies on malware detection. In the early days, signature-based detection method was widely used. This method works fast and efficiently against the known malware, but does not perform well against the zero-day malware. With the exponential increase in malware, the use of machine learning techniques became popular and became a standard for the development of new models. Fig. 5 graphically describes how our architecture works.

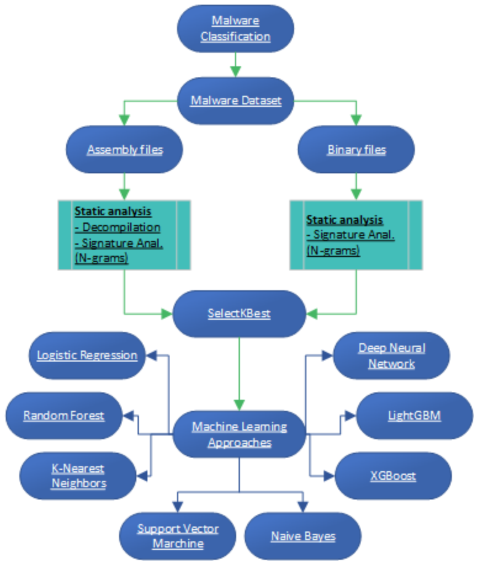


Fig. 5: Malware analysis architecture.

In this section, we perform the evaluation of machine learning and deep learning models using the 892 features selected with SelectKBest and normalized in the section V-C.

For the measurement of our malware classifier, we divided the dataset into three sets: (1) the training set, (2) the validation set, and (3) the test set, each containing 70%, 15%, and 15% of malware samples, respectively. In Fig. 6, we observe the malware classification approach through the application of machine learning and DNN algorithms.

Regarding the deep neural network used for malware classification, it makes use of a Keras sequential model, which has (1) an input layer of 892 neurons for the input of features; (2) four dense layers with dimensions of 512, 256, 128, and 64 neurons with relu activation function; (3) four intermediate layers of BatchNormalization to normalize the output, improve the stability and convergence of the model, and four intermediate layers of Dropout to avoid overfitting with a value of 0.3; (4) The last dense layer has 9 neurons and uses the softmax activation function, generating a probability distribution over 9 malware classes. With this neural network, 100% precision was achieved with a log loss of 0.00153 for the validation and 0.00241 for the test.

Table VI describes the results of the machine learning approach executed with optimization by hyperparameterization in all the models.

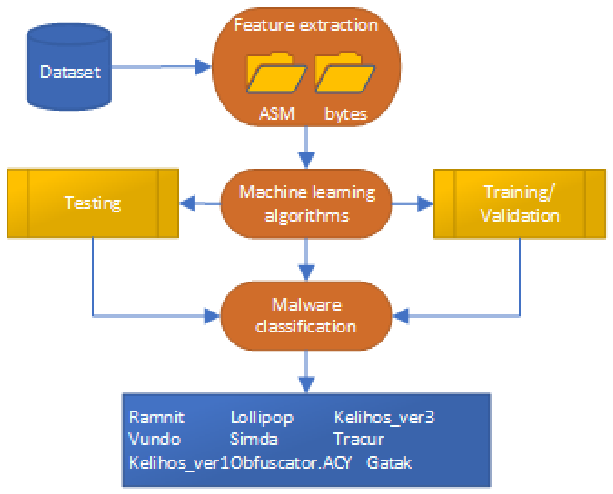


Fig. 6: Machine learning-based classification approach.

TABLE VI: Malware classification results by model, precision, and log loss.

Model	Val. Accu.	Val. Logloss	Test Accu.	Test Logloss
Logistic Regre.	94.11%	0.47543	94.24%	0.48052
Random Forest	99.57%	0.02437	99.45%	0.02502
KNN	100.00%	0.00606	100.00%	0.00605
SVM	91.53%	0.25637	91.60%	0.25405
Naive Bayes	100.00%	0.00592	99.94%	0.01025
XGBoost	100.00%	0.00655	100.00%	0.00669
LightGBM	100.00%	**0.00118**	100.00%	**0.00125**
DNN	100.00%	**0.00153**	100.00%	**0.00241**

LightGBM is a decision tree-based machine learning algorithm that has been shown to be effective in malware classification, for example [9]. According to the results obtained in Table VI, LightGBM achieves 100% accuracy with a 0.00125 prediction probability error for the malware class.

A log loss value of 0.00125 indicates that the model has excellent predictive power, as it is making accurate predictions with very little error. Generally, a log loss value below 0.1 is considered to indicate a very accurate model, while a value above 2.0 indicates a very inaccurate model.

The function matrix is presented below in Fig. 7. As can be seen, the entire main diagonal of the confusion matrix has 100% value. This means that the model has managed to classify each sample in the correct class without any error.

LightGBM is considered a lightweight and fast model due to its leaf-based tree algorithm, random data sampling, memory usage optimization, and its ability to be highly parallelizable. These characteristics are observed in the training time (∼487 ms), the validation and testing time (∼127 ms), and the size of the model (2.4 MB). These results make LightGBM an excellent choice for problems with large data sets and limited resources compared to other models such as deep neural networks, XGBoost, or Naive Bayes.

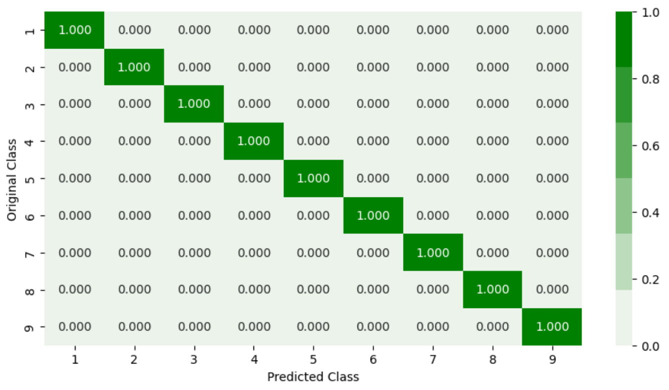


Fig. 7: Confusion matrix of malware classification by the LightGBM model.

F. Limitation

The results obtained in Section V-E demonstrate the effectiveness and efficiency of the process for malware classification. However, the system presents the following deficiencies that must be improved for its application:

i. Fluctuation in classification accuracy: Splitting the data set into three training sets does not guarantee the generalization of the model for malware classification. It is suggested to use k-fold cross-validation to evaluate its generalization capacity and obtain more precise results.
ii. Feature extension: You can add functions, procedures, segments, and other keywords extracted from ASM files. Generalization will further improve the model, but you should avoid introducing too many features to avoid overfitting.
iii. This model serves to classify malware into the nine known classes or families. It does not predict whether a file is malignant or benign. However, with a few modifications and a new database, the model can be modified and the expected results obtained.
iv. Given the characteristics of Microsoft Big 2015, it is not possible to combine the dataset with other more current ones since it uses paid tools such as Ida Pro to generate the ASM files and modifies the binary files to sterilize them (the procedure is not specified). In such a way that new malware families cannot be included.

VI. Conclusions and Future Work

Malware developers and security researchers are in a constant race to get ahead and outperform their opponents. In this constant race, malware developers are always looking for ways to evade detection, while security researchers are working on new techniques and tools to detect, classify, and prevent attacks. In the constant struggle that requires resources, skills, and knowledge to keep computer systems and networks secure.

The growing threat of cyberattacks and the evolution of malware with advanced protection techniques constantly drive researchers to develop new approaches to malware detection, classification, and removal.

The use of machine learning and deep learning techniques for malware classification enables higher accuracy in malware detection, which can be critical for early detection and prevention of cyberattacks. These models can analyze large amounts of data and detect hidden patterns that may not be detected by traditional malware detection methods.

The present investigation describes the process of classifying 10,868 pieces of malware into nine families. The data extraction allowed for the reduction of 892 features (1.1%) through the SelectKBest technique. Eight classification algorithms were applied (Logistic Regression, Random Forest, K Neasrest Neighbor, Support Vector Machine, Naive Bayes, XGBoost, LightGBM, and Deep Neural Network).

The results suggest that the LightGBM model is very accurate and reliable to classify malware samples into nine specific classes, obtaining 100% accuracy with a log loss of 0.00118 in the validation and 0.00126 in the test, improving the result presented by the competition. from the Microsoft Malware Classification Challenge, where the first three places scored 0.00283, 0.00324, and 0.00396.

The application of deep neural networks generated very satisfactory results, obtaining an accuracy of 100% with a log loss of 0.00153 in the validation and 0.00241 in the test.

As future works:

1. The architecture in Fig. 6 can be implemented in convolutional neural networks (CNNs).
2. It is also possible to apply this process to malware in the wild for its detection and classification.
3. Given the features of Microsoft Big 2015, it is suggested to take the following approaches: 1) Develop a procedure for extracting malware features using static analysis; 2) focus on the analysis of software malware in larger and freely accessible databases.

Acknowledgment

Thank God for allowing me to continue contributing to the development of malware analysis research from an artificial intelligence perspective. To Professor Paulo Lício de Geus for guiding me in this new step, and my family for supporting me.

References

[1] W. Wang, R. Sun, T. Dong, S. Li, M. Xue, G. Tyson, and H. Zhu, “Exposing weaknesses of malware detectors with explainability-guided evasion attacks,” *arXiv preprint arXiv:2111.10085*, 2021.

[2] M. Botacin, F. Ceschin, P. De Geus, and A. Grégio, “We need to talk about antiviruses: challenges & pitfalls of av evaluations,” *Computers & Security*, vol. 95, p. 101859, 2020.

[3] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, “A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019.

[4] C. Guarnieri, A. Tanasi, J. Bremer, M. Schloesser, K. Houtman, R. Zutphen, and B. Graaff, “Cuckoo sandbox-automated malware analysis,” *URL: https://cuckoosandbox. org*, 2021.

[5] O. Boyarchuck, S. Mariani, S. Ortolani, and G. Vigna, “Keeping up with the emotets: Tracking a multi-infrastructure botnet,” *Digital Threats: Research and Practice*, 2023.

[6] R. Tahir, “A study on malware and malware detection techniques,” *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.

[7] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” *arXiv preprint arXiv:1802.10135*, 2018.

[8] M. F. Botacin, P. L. de Geus, and A. R. A. Grégio, “The other guys: automated analysis of marginalized malware,” *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 87–98, 2018.

[9] H. S. Anderson and P. Roth, “Ember: an open dataset for training static pe malware machine learning models,” *arXiv preprint arXiv:1804.04637*, 2018.

[10] 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.

[11] D. Gibert, C. Mateu, J. Planes, and J. Marques-Silva, “Auditing static machine learning anti-malware tools against metamorphic attacks,” *Computers & Security*, vol. 102, p. 102159, 2021.

[12] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, and S. Venkatraman, “Robust intelligent malware detection using deep learning,” *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[13] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, “Malinsight: A systematic profiling based malware detection framework,” *Journal of Network and Computer Applications*, vol. 125, pp. 236–250, 2019.

[14] Ö. A. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[15] M. Pietrek, “An in-depth look into the win32 portable executable file format, part 2,” *MSDN Magazine, March*, 2002.

[16] ——, “Peering inside the pe: a tour of the win32 (r) portable executable file format,” *Microsoft Systems Journal-US Edition*, vol. 9, no. 3, pp. 15–38, 1994.

[17] R. Sihwail, K. Omar, and K. Z. Ariffin, “A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis,” *Int. J. Adv. Sci. Eng. Inf. Technol*, vol. 8, no. 4-2, pp. 1662–1671, 2018.

[18] D. Ucci, L. Aniello, and R. Baldoni, “Survey of machine learning techniques for malware analysis,” *Computers & Security*, vol. 81, pp. 123–147, 2019.

[19] 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.

[20] A. Souri and R. Hosseini, “A state-of-the-art survey of malware detection approaches using data mining techniques,” *Human-centric Computing and Information Sciences*, vol. 8, no. 1, pp. 1–22, 2018.

[21] H. El Merabet and A. Hajraoui, “A survey of malware detection techniques based on machine learning,” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 1, 2019.