

Comparative Analysis of Input Image Characteristics in Convolutional Neural Network-based Signature Detection

Matej ADAMEC, Michal TURCANIK

Dept. of Informatics, Armed Forces Academy of gen. M. R. Štefánik, Demänovská cesta 393, 031 01 Liptovský Mikuláš, Slovakia

{matej.adamec, michal.turcanik}@aos.sk

Submitted January 8, 2025 / Accepted February 13, 2025 / Online first May 13, 2025

Abstract. *The detection of malware represents a primary concern in contemporary computer security and is therefore imperative for the protection of systems and data integrity. This research presents an innovative approach to comparing diverse input image formats with the objective of identifying the optimal methodology for detecting specific malware-related signatures using convolutional neural networks (CNN), which have been specifically developed by the authors for this purpose. Subsequently, machine code instructions are generated and then converted into four distinct image format options. The four image formats, namely 1xN fixed, 1xN scalable, NxN fixed, and NxN scalable, are subsequently employed for the training of the CNN. The study assesses the formats in question in terms of training time, accuracy, and computational complexity. The results demonstrate that the NxN scalable format exhibits the highest accuracy with accelerated training times in comparison to other formats. Furthermore, the scalable format necessitates only 25% of the original pixel count for a 96% classification success rate. The utilization of the NxN scalable format for machine code instruction representation results in enhanced accuracy, accelerated training, and a considerable reduction in pixel usage, indicating a promising avenue for optimizing the efficiency of malware detection.*

Keywords

Signature detection, CNN malware detection, machine code visualization, static analysis, interpolation

1. Introduction

Malware represents a persistent threat to networked systems, manifesting in a multitude of forms and necessitating the continuous evolution of detection techniques [1]. Despite advancements in the field of cybersecurity, the profit motives of cybercriminals drive the rapid proliferation of sophisticated and stealthy malware variants, which present a significant challenge to traditional detection methods [2].

Convolutional neural networks (CNNs) are particularly adept at image recognition tasks, utilizing their features for

both the visualization and classification of malware. The objective of this study is to enhance the classification of malware signatures by selecting the most appropriate input image format in an innovative manner.

Recent studies have demonstrated the efficacy of innovative approaches that leverage deep learning architectures, such as Convolutional Neural Networks (CNNs) and ensemble learning, in combating evolving threats [3–8]. These approaches demonstrate exceptional accuracy and resilience, thereby underscoring the efficacy of AI methodologies in the context of malware detection.

In study [3] authors employ CNN architectures for feature extraction, using ReLU dense layers, PCA, and SVD as feature selection techniques combined with machine learning classifiers (SVM, K-NN, RF) and ensemble methods, achieving significant performance improvements with PCA, including a test accuracy of 99.36% and demonstrated resilience to polymorphic malware.

A new approach is proposed in [4] for multi-classification on imbalanced data sets. It addresses class imbalance and enhances the accurate detection of multi-class anomalies. The approach outperformed XGBoost, a CNN classifier, and six ensemble classifiers.

In article [5] the authors put forth a semantic-based intelligent malware model for the identification of malicious software within the Windows system. The proposed BERT+ConvLSTM-AM(CNN) method demonstrated an accuracy of 98.81%, thereby outperforming all baseline models [5].

The deployment of autonomous systems is of critical importance for the security of networked systems. However, strategies such as centralized or distributed detection entail a trade-off in performance. The study demonstrated that a convolutional neural network (CNN) with long short-term memory (LSTM) achieved 95.5% accuracy in the detection of internet of things (IoT) malware, outperforming existing methods [6].

A DL-AMDet architecture utilizing convolutional neural network (CNN) and bidirectional long short-term memory (BiLSTM) autoencoders is proposed for the efficient and robust detection of malware in the Android

operating system, even in the absence of an internet connection. The DL-AMDet architecture demonstrated a detection accuracy of 99.935 in terms of F-score [7].

Additionally, the authors of [9] present a distributed CNN model for malware classification, which adapts to enhance performance by examining supplementary file sections and scoring functions [8].

The aforementioned methodologies are implemented in conjunction with CNNs, augmented by alternative AI methodologies. However, none of these studies address the optimal input image format. In particular, our investigation will examine the most suitable image formats for CNN utilization.

This paper presents a comprehensive investigation of malware detection techniques, with a particular emphasis on signature-based detection utilizing convolutional neural networks (CNNs). We begin by providing an overview of the various analytical approaches employed in the field of malware research, including static, dynamic, and hybrid analysis techniques. Subsequently, we examine the complexities of signature-based detection techniques, emphasizing the significance of data preprocessing and image format selection. The following sections present our methodology for preprocessing machine code into suitable image representations and outline the various image formats considered. We then establish a uniform training environment to facilitate fair comparison across different scenarios. Finally, we assess and determine the most suitable image format for robust signature detection using CNNs. Through this structured approach, we aim to make a significant contribution to the advancement of malware detection methodologies.

This approach is distinctive in its conversion of machine code instructions into visual representations and format selection, utilizing convolutional neural networks for malware detection, thereby providing a novel perspective on signature detection within the cybersecurity domain.

2. Malware Analysis

Malware analysis is the study of malware behavior. The goal of malware analysis is to understand how malware works and how to detect and remove it. It involves analyzing the suspicious binary in a secure environment to determine its characteristics and functions to develop better defenses to protect an organization's network [9].

In the past, it was optional to define these two terms. The term "detection" refers to distinguishing whether a particular file is malicious. Analysis, on the other hand, refers to understanding how the malware in question works. Nowadays, however, this dividing line is blurred. This is because the role of automated analysis tools such as sandboxes has now expanded. For manual analysis, it means understanding malware behavior, and for automated analysis, it can additionally mean detection [10].

Over the years, several malware analysis techniques have been proposed by various researchers to detect and

classify malware in general. Malware analysis methods can be divided into static and dynamic analysis. The detection techniques can be characterized based on the executable features used for detection and their classification approach [11].

2.1 Static Analysis

A simple static analysis examines the ensemble without examining machine instructions. However, simple static analysis can confirm whether a file is malicious, providing information about its functionality that can be used to create simple signatures of the malicious file [1], [12].

Advanced static analysis is an in-depth examination of the code of a potentially malicious program. The result is single-machine instructions that accurately describe the operation of the file. This approach is more time-consuming and complex but results in a complete file analysis [1], [12].

Most applications are compiled and stored in hexadecimal format. The compiled program loses valuable information that an analyst could use, such as variable names, etc.

2.2 Dynamic Analysis

Malware often has various code obfuscation techniques that make static analysis extremely difficult. Dynamic analysis can overcome these code obfuscation techniques by running the malware [2].

Dynamic analysis, however, requires a secure environment in which we can dynamically analyze the process without inadvertently falling victim to the potentially malicious file being examined. The principle is to monitor file system registry calls, network traffic, and process and thread activity [1, 12, 13].

2.3 Hybrid Analysis

Hybrid analysis combines the techniques of static and dynamic analysis so that it takes advantage of both approaches. The software observes using static code analysis, checking suspicious areas of the code and then running the program in a virtual environment to observe its actual behavior. The different techniques help each other in extracting artefacts [14].

3. Malware Detection

A program designed to detect malicious activity is known as a malware detector, many times referred to as antivirus software. The general function of malware detection can be defined as follows:

$$D(p) = \begin{cases} \text{Malicious} & \text{if } (p) \text{ contains malicious code} \\ \text{Benign} & \text{other cases} \end{cases} \quad (1)$$

where $D(p)$ is a function that can verify whether an application or program (p) is either benign or malicious. In some

cases, $D(p)$ cannot correctly decide whether a program is malicious or harmless. This case is adjusted in (2):

$$D(p) = \begin{cases} \text{Malicious} & \text{if } (p) \text{ contains malicious code} \\ \text{Benign} & \text{if } (p) \text{ is standard application} \\ \text{Undecidable} & \text{if } D(p) \text{ cannot determine } p \end{cases} \quad (2)$$

The undecidable state may be due to a new type of malware or simply the detector's inability to detect that type of malware. Therefore, we have added an undecidable state to the function, which covers the set of unclassifiable programs. Mostly zero-day vulnerabilities fall into this set [14].

3.1 Signature-Based Detection

When malware is created, a unique sequence of bits, commonly known as a signature, is inadvertently inserted into its code, which can later be used to identify the malware and determine its class. This detection technique is called string/pattern scanning or simply matching. Signature detection can work statically, dynamically or hybrid [14], [15].

3.2 Heuristic-Based Detection

Heuristic-based detection identifies or distinguishes between normal and abnormal system behavior and can use the principles of signature detection and behavior detection to identify and resolve known or unknown malware attack attempts [16].

The heuristic detection process consists of two steps. In the first step, observation of the system takes place. In this step, the security of each running application is guaranteed, and essential information is recorded, which can be verified and checked in the second step. In the second step, the differences created by the running applications are actively monitored. These changes may reveal malware [14], [15].

3.3 Specification-Based Detection

A specification-based technique monitors applications according to their specification and checks for normal and abnormal behavior. This technique is derived from the heuristics-based approach. Still, the main difference is that the heuristics technique uses artificial intelligence and machine learning methods to determine malicious and benign software. In contrast, specificity detection is based on analyzing the behavior described in the system specification. This method is a manual comparison of the normal activities of a particular software [14].

Signature-based detection is efficient at identifying known malware with low computing power but cannot detect unknown threats without pre-defined signatures. Heuristic-based detection can identify both known and new malware but requires frequent updates, consumes more resources, and has a high rate of false positives. Specification-based detection effectively identifies known and unknown malware with low false positives but struggles with high false negatives and is time-consuming to define

Detection method	Advantages	Disadvantages
Signature-based detection	- Easily detects known malware. - Uses significantly less computing power than other techniques.	- Fails to detect unknown malware (for which no signature has been defined).
Heuristic-based detection	- Detects known and unknown new malware.	- New malware characteristics need to be updated. - More time and disk space consumption. - False positives is high.
Specification-based detection	- Detects known and unknown new malware. - Low level of false positives.	- Level of false negatives is high. - Ineffective in detecting new malware. - Defining specifics is time-consuming.

Tab. 1. Comparison of detection methods.

specifics. The advantages and disadvantages are summarised in Tab. 1.

4. Data Sets Setup for CNN

The research was conducted in a computational environment powered by a DELL XPS 8930 equipped with an Intel Core i7-8700 CPU, 16GB DDR4 RAM, M.2 PCIe SSD 256GB combined with a 2000GB HDD and an NVIDIA GeForce GTX 1080 GPU.

In our research, we aim to assess the effectiveness of different image formats, namely $N \times N$ fixed, $N \times N$ scalable, $1 \times N$ fixed, and $1 \times N$ scalable, in identifying specific malware-related signatures. Instead of using actual malware samples, we generate source codes and signatures using our proprietary generator. Our objective is to determine which image format best represents the characteristic patterns of real malware signatures. We have discussed source code generation in previous work from which we will obtain necessary information [17].

In our previous work [17], we defined what is signature, infected and plain code. A plain code is a randomly generated sequence of machine instructions without any intentionally inserted sequence of instructions (signature). A signature is a randomly generated sequence of instructions which is then inserted into the plain code at a random position. To verify the ability of the method using CNN to recognize any kind of text - malware signatures. This treats the plain code as infected and marks it according to the corresponding signature it contains. The signatures and the plain code are randomly generated from the instruction set for the 8086 microprocessor. Subsequently, these instructions are converted into an image.

4.1 Machine Code

The central processing unit (CPU) executes instructions (also called machine instructions). The instructions that the CPU executes are stored in memory as a sequence of bytes. During the execution of the instructions, the required

data (which is also stored as a sequence of bytes) is retrieved from memory. The CPU itself contains memory in its processor, which is called a register memory. The registers are used to store the values loaded from memory during execution [9].

Each processor has a set of instructions that it can execute. The instructions that the CPU executes make up the CPU's instruction set. These machine instructions are stored in memory as a sequence of bytes that the CPU reads, interprets, and executes. A compiler is a program that translates programs written in a programming language (such as C or C++) into machine language [3].

Our approach utilizes machine code instructions as the primary resource for generating images used in classification. The example code for generating source codes is shown in Fig. 1.

Table 2 illustrates the manner in which an image is constructed from instructions, with each instance of the letter *i* representing the RGB value of a particular machine instruction.

4.2 Image Creation Process

A program's source code is made up entirely of instructions. We convert this source code to a visual representation before training a convolutional neural network to recognize code by image. Consequently, we give each machine code instruction a special color from the RGB spectrum. It offers us more than 16 million possible options. This fully satisfies our needs. Instructions are created for the image one at a time, going from left to right from top to bottom. The example code is shown in Fig. 2.

```
for file in filecount :
    final = "source " + str(file) + ".txt"
    list = random.choices(machine_code_source, k=size)
    filefinal = open("source_code/"+final, "a")
    filefinal.writelines(list)
    filefinal.close()
```

Fig. 1. Machine instruction generation code.

←Width of Image (pixels) →				
↑ Height of Image (pixels) ←	i_1	i_2	i_3	i_4
	i_5	i_6	i_7	i_8
	i_9	i_{10}	i_{11}	i_{12}
	i_{13}	i_{14}	i_{15}	i_n

Tab. 2. Image layout.

```
for line in file:
    if "[" in str(fcolor(line)):
        with open('not.txt', 'w') as f:
            pixel.append(fcolor(line)); c = c + 1
            if line == c:
                pixelf.append(pixel); c = 0; a = a + 1
                pixel = []
            if a == height:
                arr = numpy.array(pixelf)
                arrayfinal = np.array(arr, dtype=np.uint8)
                filename = str(file) + ".bmp"
                new_image = Image.fromarray(arrayfinal)
                new_image.save(newpath + "/" + filename)
                a = 0
                pixel = []
                pixelf = []
```



Fig. 2. Image generation code.

4.3 Image Size


The first consideration is determining the appropriate image resolution. Let's explore the following options. In both fixed and scalable formats, missing instructions (pixels) are filled with a NOP operation, represented as a white pixel. After careful evaluation, we identified two primary approaches: maintaining a fixed image size or adjusting it. Each approach is further divided into 1xN and NxN formats. We consider the Nx1 format equivalent to the 1xN format. The value of N is set by the user to best fit the application:

1 x N (Fixed) - The source code image in this possibility will have a defined width and height. Let the image's resolution be 1x10. In other words, 10 instructions are contained in one image. For illustration as shown in Tab. 3, two photos one with 10 and second with 10 visualized instructions will make up a source code file with 20 instructions. N- is user-defined.


1 x N (Scalable) - This option, regardless of image size, transfers the whole source code to a single image file as shown in Tab. 4. The image is sent through a scaling filter to achieve the uniform input size before being fed into the neural network. The definition of N is the number of instructions in one source of code.

Fixed image height and width (1x10)	
Number of instructions / image count	Images – Grey area around is border not image
20 / 2	 <p>Fig. 3. 1xN Fixed image I</p>
	 <p>Fig. 4. 1xN Fixed image II</p>

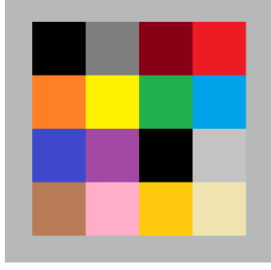
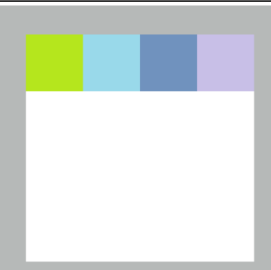
Tab. 3. 1 x N fixed.

Scalable image width and fixed height	
Number of instructions / image count	Images
20 / 1	 <p>Fig. 5. 1 x N Scalable image</p>

Tab. 4. 1 x N scalable.

Scalable image width and height	
Number of instructions / image count	Images – Grey area around is border not image
20 / 1	 <p>Fig. 6. Scalable image width and height</p>

Tab. 5. N x N scalable.

Fixed image width and fixed height (4x4)	
Number of instructions / image count	Images – Grey area around is border not image
20 / 2	 <p>Fig. 7. NxN Fixed image I</p>
	 <p>Fig. 8. NxN Fixed image II</p>

Tab. 6. N x N fixed.

N x N (Scalable) - The size of N is calculated so that all instructions fit in the image and there are the least number of unused fields. Thus, we calculate N as the square root of the total number of source code instructions rounded up to

an integer. Thus, if we have a source code of size 20 instructions, we have calculated N to be 5. Thus, the resulting image will be 5x5 as shown in Tab. 5. The image will then be filtered to a final unite resolution before entering the neural network. N is defined as square root rounded up to integer.

N x N (Fixed) - The width and height of the source code picture are fixed. For illustration we decided on a size of 4x4 for testing. This means that a single image has 16 pixels and can therefore include 16 instructions. Then two pictures containing a source code with 20 instructions will be generated as shown in Tab. 6. N is user-defined.

4.4 Image Resize Tool (Interpolation)

In order to avoid any potential loss of pixels, a tool from the Cv2 library in the Python environment is utilized for image resizing. This technique is also known as interpolation. The extra pixels in the new image are computed via interpolation. We use the Inter Cubic method. Inter cubic does bicubic interpolation across a neighborhood of 4 x 4 pixels [18]. We use interpolation, only in the scalable format. The fixed format is not rescaled. Python code for interpolation is shown in Fig. 9.

4.5 CNN Training Setup

Four sets of codes were developed to assist in the selection of the most appropriate image format. The first set, designated as "clear," lacks an intended order of instructions. In contrast, each signature in the remaining three sets is precisely one. These sets will remain constant throughout the duration of the research. The complete set is presented in Tab. 7.

We will select an existing convolutional neural network model and adapt it to our needs. Our initial focus is on establishing an appropriate image format (NxN / 1xN – Scalable/Fixed). This involves the conversion of a program into machine code and subsequently into an image. Even with a highly optimized model, it would not perform well with

```
def rescale(source,x,y):
    for subdir, dirs, files in os.walk(source):
        for file in files:
            filename = subdir + os.sep + file
            filepath="rescaled/"+subdir + os.sep
            if not os.path.exists(filepath):
                os.makedirs(filepath)
            img=cv2.imread(filename)
            res = cv2.resize(img, dsize=(x, y),
            interpolation=cv2.INTER_CUBIC)
            rescaled = Image.fromarray(res)
```

Fig. 9. Python code for image interpolation.

Code group	Shortcut	Number of instructions in one code	Number of codes	Signature length
Clear	Clear	300	1000	0
Signature 0	S_0	300	4000	100
Signature 1	S_1	300	4000	100
Signature 2	S_2	300	4000	100

Tab. 7. Complete dataset.

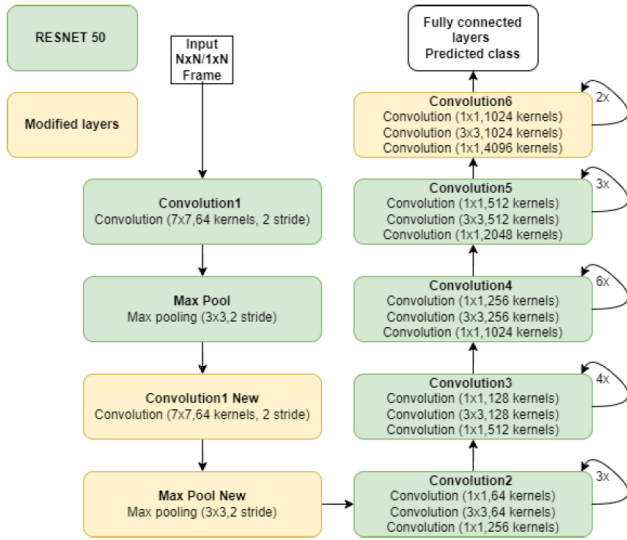


Fig. 10. Topology of the modified CNN.

unsuitable inputs. After conducting research, we decided to adopt the modified ResNet50 model as our choice following up on previous research [17].

The ResNet architecture adheres to two key design tenets. Initially, there are the same number of filters in each layer regardless of the output feature map's size. Second, even though the size of the feature map is cut in half, it contains twice as many filters to maintain the time complexity of each layer [19].

The original Resnet50 model is modified as shown in Fig. 10. We duplicated first convolution and max pooling layers. Also before fully connected layers we inserted two more layers named Convolution6. In summary we added 7 more layers. After training original and modified ResNet50 on our data set, we consider that our modified CNN achieves better classification results than the original ResNet-50 architecture.

Tab. 8 presents the training options for training a modified CNN model. These options define the parameters and settings that govern the training process and influence the model's learning behavior. Training options are set as given in Tab. 8.

Image format and option selection will undertake a comparative analysis of the aforementioned alternatives,

Solver	
Solver:	Stochastic gradient descent with momentum
Initial learn rate:	0.01
Basic	
Validation frequency:	50
Max epochs:	30
Minimal batch size:	128
Shuffle:	Every epoch
Drop factor:	0.1
Momentum:	0.9
Training data:	80%
Validation data:	20%

Tab. 8. Modified CNN training options.

with a particular focus on the selection between the NxN and 1xN formats. In addition to accuracy, a crucial component that will be observed is the computational complexity reflected by the training time. The subsequent section will delve deeper into the selection between scalable and fixed formats, evaluating the potential drawbacks of both methods.

5. Image Format and Option Selection

The initial decision is to select one of two formats: NxN or 1xN. Subsequently, the selected format is then further refined by choosing between Scalable and Fixed option. The decision will be based on the time spent training and accuracy. In the context of time consuming hypotheses, it can be posited that a reduction in time will result in a corresponding decrease in the computational intensity of the process. Similarly, in the context of accuracy hypotheses, it can be proposed that an increase in accuracy will lead to a superior outcome.

Time consuming hypotheses are:

- H_0 -NxN is less time consuming.
 $([NxN \text{ fixed elapsed time} + NxN \text{ scalable elapsed time}]) < ([1xN \text{ fixed elapsed time} + 1xN \text{ scalable elapsed time}])$.
- H_1 -1xN is less time consuming.
 $([1xN \text{ fixed elapsed time} + 1xN \text{ scalable elapsed time}]) < ([NxN \text{ fixed elapsed time} + NxN \text{ scalable elapsed time}])$.
- H_2 - NxN and 1xN are comparatively time consuming
 $([NxN \text{ fixed elapsed time} + NxN \text{ scalable elapsed time}]) \approx ([1xN \text{ fixed elapsed time} + 1xN \text{ scalable elapsed time}])$.

Accuracy hypotheses are:

- H_0 -[Format (NxN/1xN)] Fixed option is more accurate.
 $([Format] \text{ fixed accuracy}) < ([Format] \text{ scalable accuracy})$.
- H_1 -[Format] Scalable option is more accurate.
 $([Format] \text{ fixed accuracy}) > ([Format] \text{ scalable accuracy})$.
- H_2 -[Format] Scalable and [Format] Fixed options are comparatively accurate.
 $([Format] \text{ fixed accuracy}) \approx ([Format] \text{ scalable accuracy})$.

We built a homogeneous computing environment for training. Such that, the analysis of time consumption was unaffected by any background processes. By homogeneous computing environment we mean that all experiments had the same hardware capacities. No other processes or applications degraded the performance of the devices on which the capability test was run during testing. Table 9 shows the

number of samples selected from the total dataset. The tested formats are as follows:

NxN fixed - option can discover known signatures quickly and efficiently. A notable drawback is the possibility that, similar to the 1xN fixed, the signature may be split into two images during the classification of the program's source code. Convolutional neural network won't be able to classify source codes properly.

1xN Fixed - Making an image from source code as a line of pixels is the simplest method. As a result, we prevent splitting the potential bit sequence into a separate row of pixels, which can occur with NxN types. The drawback of this strategy against NxN is the wider image, which increases processing complexity. Since the signature can be split into two images because of the fixed width and is therefore hidden, there may be an issue with the classification of the images.

NxN Scalable - One of the finest possibilities looks to be NxN scalable. And the reason for that is that each code always generates a single image. This means that, unlike with the fixed option, there can never be a situation where the signature is split into many images. The drawback of reducing the image resolution is that the signature loses its individuality.

1xN Scalable - The benefit of the last choice, 1xN scalable as well as 1xN fixed, is that the signature is not divided into many lines of pixels. Another benefit of the scalable option is that it always produces a single image during classification, ensuring that the signature will remain intact.

5.1 NxN vs 1xN format (Time Consuming Hypothesis)

The modified CNN model was able to accurately classify 100% of the photos in scalable resolutions, just like it would in fixed resolution. All four possibilities have demonstrated their ability to accurately recognize signatures.

Although training the model multiple times is a key component of the decision-making process in the following section, we only chose to proceed with the NxN Fixed and NxN Scalable options due to their much shorter training durations than the 1xN resolutions. According to Tab. 10 the

Train/Validate Dataset			
Code	Number of instructions in one code	Number of codes	Signature length
Clear	300	1500	0
Signature 0	300	1500	100
Signature 1	300	1500	100
Signature 2	300	1500	100
Total number train/val set: 6000			
Test Dataset			
Clear	300	25	0
Signature 0	300	15	100
Signature 1	300	20	100
Signature 2	300	40	100
Total number test set: 100			

Tab. 9. Train/Validate/Test set I.

Image format	Format option	Accuracy	Elapsed time	Total time (per format)
NxN	Scalable	100%	601 min.	1199 minutes
	Fixed	100%	598 min.	
1xN	Scalable	100%	3403 min.	6704 minutes
	Fixed	100%	3301 min.	

Tab. 10. NxN vs 1xN format.

null hypothesis (H_0) was confirmed, while the alternative hypotheses (H_1 and H_2) were rejected. This indicates that NxN is a more time-efficient approach, with a reduced computational complexity. This is due to the fact that the sum of NxN training times is 1,199 minutes, whereas the sum of 1xN training times is equal to 6,704 minutes.

5.2 NxN Scalable vs. NxN Fixed (Accuracy Hypothesis)

The number of source codes was reduced in order to accelerate the training time required to determine whether a scalable or fixed option would be more beneficial. The set of images used in this experiment is displayed in Tab. 11. The training set and validation set are known for 80% and 20% of the total number of images, the test set consists of 100 images correspondingly, according to the table.

The training of neural networks will be conducted in three rounds, with a total of six instances. Due to the insufficient resolution of the image, the number of images in the fixed option will increase with each instance where the resolution is inadequate for displaying the entirety of the code instructions. The discrepancy in image count is illustrated in Tab. 12.

Train/Validate Dataset			
Code	Number of instructions in one code	Number of codes	Signature length
Clear	300	1500	0
Signature 0	300	1500	100
Signature 1	300	1500	100
Signature 2	300	1500	100
Total number train/val set: 6000			
Test Dataset			
Clear	300	25	0
Signature 0	300	25	100
Signature 1	300	25	100
Signature 2	300	25	100
Total number test set: 100			

Tab. 11. Train/Validate/Test set II.

N.	NxN image format	Instruction count	Final size after rescaling	Image count
1	Fixed	300	16x16	12000
	Scalable	300	16x16	6000
2	Fixed	300	14x14	12000
	Scalable	300	14x14	6000
3	Fixed	300	9x9	24000
	Scalable	300	9x9	6000

Tab. 12. Image count.

The reason why the total number of images in the fixed image format is the same in the first and second rounds is that at a resolution of 16x16 we cover 256 instructions and to display the entire code (300 instructions) we need 2 images. This is twice the number of images compared to the scalable format. Similarly, round 2, where the resolution is 14x14, will cover 196 instructions. We will also need 2 images per code (300 instructions). So round one and round two are the same in number of images but different in resolution. The third round already has a resolution of 9x9, so it covers 81 instructions. This means that we need 4 images to display 300 instructions, which is four times as much for the fixed version as for the scalable version.

Table 13 presents a comparison of individual options against the appropriate 18x18 format. We consider this format appropriate only for this experiment because we can display all the instructions in one image without changing the image's size or needing to divide it into multiple images.

Once all six neural networks were properly trained, the test set was created. And the key reason for this is to prevent mistakenly using the test set during training while handling a lot of images. There are 100 images in the test set. One quarter for each class, which is 25 images per class. The number of instructions in a single code and the length of the signature match those of the training set.

Notwithstanding the 75% loss of pixels, Scalable Option was able to correctly classify 97% of the images. As illustrated in Tab. 14 results corroborate Hypothesis 1, indicating that Scalable Option is more accurate.

Table 15 provides a comparison of different methods used for signature processing. Each method is evaluated based on its advantages and disadvantages, highlighting their key characteristics.

By understanding the advantages and disadvantages of each method, researchers and practitioners can make informed decisions about the most suitable approach for signature processing based on their specific requirements and priorities.

Image format	Image size	Percentual image count growth	Instruction loss (Pixel loss)
Fixed	16x16	100%	0%
	14x14	100%	0%
	9x9	400%	0%
Scalable	16x16	0%	20.9%
	14x14	0%	60.5%
	9x9	0%	75%

Tab. 13. Image properties.

Image format	Size	Classified/Absolute	Accuracy
Scalable	16x16	100/100	100%
	14x14	100/100	100%
	9x9	97/100	97%
Fixed	16x16	100/100	100%
	14x14	27/100	27%
	9x9	8/100	8%

Tab. 14. NxN vs 1xN accuracy.

Method	Advantages	Disadvantages
NxN Fixed	Fast training process.	Signature may split into multiple images. Signature may be divided into multiple pixel rows.
1xN Fixed	Signature won't be divided into multiple rows.	Lengthy training process. Signature may split into multiple images.
NxN Scalable	Fast training process. Signature will not be split into multiple images.	Resizing may affect signature individuality.
1xN Scalable	Signature won't be divided into multiple rows. Signature will not be split into multiple images.	Lengthy training process. Resizing may affect signature individuality.

Tab. 15. Summary.

6. Results

Initially, a comparison was conducted between the NxN and 1xN formats, from which it was determined that the NxN format was more suitable according to Tab. 10. It should be noted that the kernels in the 1xN options were not modified to 1D. Nevertheless, this did not impact the results, as CNN was still able to classify 100% of the images in accordance with Tab. 10. The NxN format is, on average, five times as fast as 1xN. Consequently, we proceeded with the NxN format exclusively and explored the Scalable and Fixed options in conjunction with it.

Despite a 75% reduction in pixel data, the Scalable Option demonstrated impressive resilience, achieving a 97% classification accuracy rate. As demonstrated in Tab. 14, these results confirm that the Scalable Option provides superior accuracy, even with significantly reduced visual information. This outcome suggests that the Scalable Option effectively preserves critical features needed for accurate classification, making it a highly efficient approach for image-based malware detection.

Future convolutional neural network signature recognition applications would benefit much from implementing the NxN format with scalable option. Although using only 25% of the pixels in the 9x9 format as opposed to the 18x18 version, we were still able to properly identify 97% of the signatures.

7. Conclusion

CNN's have proven themselves in a variety of security-related applications. CNN's have also been successfully used in other security domains that do not have an obvious image-based component. By treating executables as images, researchers have been able to leverage the strengths of CNNs for malware detection, classification, and analysis. CNNs are successfully applied to a combination of static and dynamic features [20]. In this research CNN created by authors

Who	Image format	Data preprocessing	Model	Accuracy	Dataset
[21]	not stated	Converted into opcode	ResNet 50	91.6 %	Public binary (Benign/Malware)
[22]	not stated	Word to vector	Custom	97.1%	Public binary (Benign/Malware)
[23]	not stated	Feature extraction from existing list of features	CNN-LSTM	99%	Public binary (Benign/Malware)
Our work	Scalable NxN	Machine code instructions	Our modified CNN	97%	Pseudorandom generated classes.

Tab. 16. Overall summary.

was used which has better results than topologies presented in the introduction of the paper.

A comparison of the different methods can be seen in Tab. 16. It is important to note the different datasets and their difficulty of detection. Freely available datasets may have truncated signatures which are easier to visualize in the code.

By testing different image formats, we have proven that the most suitable format in the malware detection problem is the NxN scalable resolution. This is because of the speed and accuracy compared to the other tested formats.

We first examined which of the 1xN or NxN options is the more suitable adept. After evaluation, we clearly proved that the scalable option is the most suitable image generation method for such an application. From the beginning, the scalable NxN format appeared to be the best and most optimal format. Which we proved at the end of the work (Tab. 15).

One of the biggest advantages of the scalable format is that we needed 25% of the original number of pixels for a 97% classification success rate. We can use this feature in real applications because the samples can be of different sizes. Another important fact is that we solved the problem of the CNN input layer, which is fixed for each trained model.

In our future work, our emphasis will be on the proper preprocessing of programs and determining the most suitable approach for obtaining machine instructions. We are considering the utilization of Ghidra in headless mode to convert a significant volume of real malware samples into machine language. This converted machine language datasets will serve as the basis for generating images in our future research.

References

- [1] SIKORSKI, M., HONIG, A. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. 1st ed. USA: No Starch Press, 2012. ISBN: 978-1593272906
- [2] HENG, Y., SONG, D. *Automatic Malware Analysis: An Emulator Based Approach*. New York (USA): Springer, 2013. ISBN: 978-1461455226
- [3] KUMAR, S., JANET, B., NEELAKANTAN, S. IMCNN: Intelligent Malware Classification using Deep Convolution Neural Networks as Transfer learning and ensemble learning in honeypot enabled organizational network. *Computer Communications*, 2024, vol. 216, p. 16–33. DOI: 10.1016/j.comcom.2023.12.036
- [4] XUE, L., ZHU, T. Hybrid resampling and weighted majority voting for multi-class anomaly detection on imbalanced malware and network traffic data. *Engineering Applications of Artificial Intelligence*, 2024, vol. 128, p. 1–19. DOI: 10.1016/j.engappai.2023.107568
- [5] LIU, J., ZHAO, Y., FENG, Y., et al. SeMalBERT: Semantic-based malware detection with bidirectional encoder representations from transformers. *Journal of Information Security and Applications*, 2024, vol. 80, p. 1–12. DOI: 10.1016/j.jisa.2023.103690
- [6] MEHRBAN, A., AHADIAN, P., *Malware Detection in IOT Systems Using Machine Learning Techniques*. 12 p. [Online] Available at: <https://arxiv.org/pdf/2312.17683>, 2024. DOI: 10.48550/arxiv.2312.17683
- [7] NASSER, A. R., HASSAN, A. M., HUMAIDI, A. J. DL-AMDet: Deep learning-based malware detector for android. *Intelligent Systems with Applications*, 2024, vol. 21, p. 1–10. DOI: 10.1016/j.iswa.2023.200318
- [8] QUERTIER, T., BARUÉ, G., *Use of Multi-CNNs for Section Analysis in Static Malware Detection*. 10 p. [Online] Available at: <https://arxiv.org/pdf/2402.04102>, 2024. DOI: 10.48550/arXiv.2402.04102
- [9] MONNAPPA, K. A. *Learning Malware Analysis: Explore the Concepts, Tools, and Techniques to Analyze and Investigate Windows Malware*. Packt Publishing, 2018. ISBN: 978-1788392501
- [10] AFIANIAN, A., NIKSEFAT, S., N., SADEGHIYAN, B., et al. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys*, 2019, vol. 52, no. 6, p. 1–28. DOI: 10.1145/3365001
- [11] VIDYARTHI, D., KUMAR, C. R. S., RAKSHIT, S., et al. Static malware analysis to identify ransomware properties. *International Journal of Computer Science Issues*, 2019, vol. 16, no. 3, p. 10–17. DOI: 10.5281/zenodo.3252963
- [12] OKTAVIANTO, D., MUHARDIANTO, I. *Cuckoo Malware Analysis*. Packt Pub Ltd, 2013. ISBN: 978-1782169239
- [13] ZHANG, S., WU, J., ZHANG, M., et al. Dynamic malware analysis based on API sequence semantic fusion. *Applied Sciences*, 2023, vol. 13, no. 11, p. 1–16. DOI: 10.3390/app13116526
- [14] TAHIR, R. A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, Pakistan, 2018, vol. 8, no. 2, p. 20–30. DOI: 10.5815/ijeme.2018.02.03
- [15] JAWAD, A. R., KHAIRONI, Y. S., AMMAR, K. A. N/A and signature analysis for malwares detection and removal. *Indian Journal of Science and Technology*, 2019, vol. 12, no. 25, p. 1–7. DOI: 10.17485/ijst/2019/v12i25/146005
- [16] HOSSAIN FARUK, M. J., SHAHRIAR, H., VALERO, M., et al. Malware detection and prevention using artificial intelligence techniques. In *2021 IEEE International Conference on Big Data (Big Data)*. Orlando (FL, USA), 2021, p. 5369–5377. DOI: 10.1109/BigData52589.2021.9671434
- [17] TURCANIK, M., ADAMEC, M. Malware signatures detection with neural networks. In *2022 New Trends in Signal Processing (NTSP)*. Liptovsky Mikulas (Slovakia), 2022, p. 1–8. DOI: 10.23919/NTSP54843.2022.9920380
- [18] ROSEBROCK, D. A. *Practical Python and OpenCV: An Introductory, Example Driven Guide to Image Processing and Computer Vision*. PyImageSearch, 2016.

- [19] ROSEBROCK, D. A. *Deep Learning for Computer Vision with Python*. PyImageSearch, 2017.
- [20] STAMP, M., ALAZAB, M., SHALAGINOV, A. *Malware Analysis Using Artificial Intelligence and Deep Learning*. 1st ed. Springer International, 2021. ISBN: 978-3030625818
- [21] ABDELKHALKI, J. E., AHMED, M. B., ABDELHAKIM, B. A. Image malware detection using deep learning. *International Journal of Communication Networks and Information Security*, 2020, vol. 12, no. 2, p. 180–189. DOI: 10.17762/ijcnis.v12i2.4600
- [22] BAKHSHINEJAD, N., HAMZEH, A. Parallel-CNN network for malware detection. *IET Information Security*, 2020, vol. 14, no. 2, p. 210–219. DOI: 10.1049/iet-ifs.2019.0159
- [23] AKHTAR, M. S., FENG, T. Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time. *Symmetry*, 2022, vol. 14, no. 11, p. 2308–2321. DOI: 10.3390/sym14112308

About the Authors ...

Matej ADAMEC was born in Žiar nad Hronom, Slovakia. He received his master's degree from the Armed Forces Academy of Gen. M.R. Štefánik in 2021. His research interests include forensic analysis and programming.

Michal TURCANIK is an Associate Professor at the Department of Informatics, Armed Forces Academy of general Milan Rastislav Štefánik in Liptovský Mikuláš. He has been teaching different courses for more than 20 years. He is an IST Panel Member of the NATO–STO organization for the Slovak Republic. His scientific research is focusing on reconfigurable logic, artificial intelligence, and computer networks.