Image-Based Malware Analysis Using Residual Capsule Network

Omaima El alaoui-Elfels, Taoufiq Gadi, and Abdelhak Bassir.

***Abstract*—Cyberattacks and malware detection software are in a perpetual race. Researchers have developed antimalware systems that can detect and classify malicious software automatically rather than manually analyzing malware files, which is time-consuming and ineffective. On the other hand, cyberattackers have developed approaches to avoid signature-based detectors used by antivirus software. In order to improve these detectors, deep learning has been applied by automating feature extraction and malicious pattern identification. Convolutional Neural Networks (CNNs) as a deep learning architecture have proved their effectiveness in malware classification. However, these networks require a large dataset for training, and cannot handle spatial relationships among features effectively; they lose position information on the exact location of a feature, which is critical for a malware file that contains a series of sections. In this paper, we exploit the concept of image classification in the field of computer vision to detect and classify malware images using the ELU Residual Capsule Network (ER-Caps), which goes beyond CNNs' shortcomings and shows a high performance on a highly imbalanced malware dataset without any enhanced training strategies. The malware files are converted into grayscale images and then processed by an advanced residual subnet to extract the complex features. The resulting feature maps are grouped into capsules and then processed by the dynamic routing algorithm to classify malware. The experimental results indicate that ER-Caps achieves an F1-score of 91.97%, 94.02%, and 98.43% on classifying VIRUS-MNIST, MMCC, and Malimg datasets, respectively, and a malware detection rate of 99.24%, outperforming other deep learning models proposed for malware analysis.**

***Index Terms*—Cyber Security, Deep learning, Malware Visualization, Malware Detection, Malware Classification, Capsule Network, Residual Blocks, Dynamic Routing, Imbalanced Dataset.**

I. Introduction

GROWING exponentially, Internet access became an important infrastructure for universities, financial institutions, business establishments, governments, and many more other sectors. This expansion is endangered by cybercriminals with malware and cyber threats. Over the last few years, the amount of malware in circulation has expanded significantly. Malicious software has hacked computer systems all around the world [1]. Thousands of malicious programs are being developed; Figure 1 shows annual statistics of malicious software attacks during the previous ten years [2].

Manuscript received May 13, 2024; revised December 18, 2024.

O.El Alaoui-Elfels is a PhD graduate from Hassan First University of Settat, Faculty of Sciences and Techniques. Mathematics, Computer Science and Engineering Laboratory, 26000 - Settat, Morocco (e-mail: elalaouielfels.fst@uhp.ac.ma).

T.Gadi is a professor in the Mathematique Department, Hassan First University of Settat, Faculty of Sciences and Techniques, Laboratory of Mathematics, Computer Science, and Engineering, 26000-Settat, Morocco (e-mail: gtaoufiq@yahoo.fr).

A.Bassir is a professor in the Mathematique Department, Hassan First University of Settat, Faculty of Sciences and Techniques, Laboratory of Mathematics, Computer Science, and Engineering, 26000-Settat, Morocco. (e-mail:abdelhak.bassir@uhp.ac.ma).

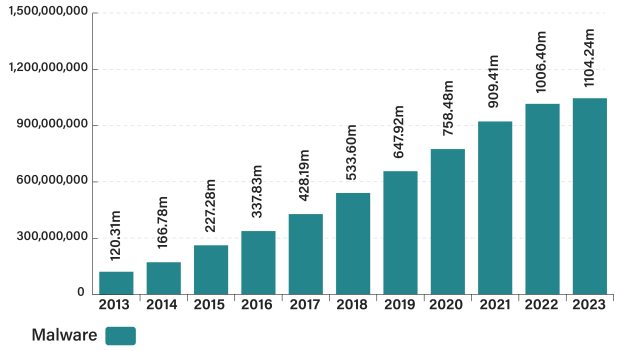

Fig. 1. Malicious software attacks for the last 10 years.

It displays that the overall quantity of active malware had escalated to surpass 1000 million in 2023, a 900% growth over the course of the last ten years. Each malware attack can cost millions of dollars to small and medium sized enterprises [3].

The process of analyzing malware attacks is continually evolving. Malware research has been conducted since the 1990s, ranging from static, dynamic, and hybrid analysis to machine learning and deep learning models [3]. The analysis of malware involves both the detection and classification of malware. Benign and malicious software can be differentiated through detection, whereas, classification involves identifying the specific malware class.

Static analysis is the process of extracting feature malware from the code of a program in order to study the attitude and configuration of a malware sample without executing it. It includes analyzing the malware's binary code and identifying its components, detecting suspicious patterns, and looking for potential malicious behavior [4]. In contrast, dynamic analysis requires a controlled environment (such as Sandbox [5], an open source, available on GitHub) in order to run the code and track its behavior. It includes monitoring network traffic, system calls, file system changes, and other actions to understand the malware behavior [4], [3]. The hybrid approaches integrate both static and dynamic features to increase the ability of malware detection [6].

This progression from static to dynamic to hybrid detection exemplifies the industry's response to malware's increasing sophistication [7]. Nevertheless, the anti-detection technique is constantly being improved. Malware employs obfuscation, polymorphism, virtual machine protection, and other methods to evade anti-killing measures. Exploiting machine learning provides adaptive pattern identification, integrating dynamic and static features for robust and scalable malware detection, minimizing evasion, and enhancing accuracy by learning patterns from enormous datasets. Several traditional

machine learning techniques, such as K-Nearest Neighbors [8], [9], Support Vector Machine [10], Naive Bayes [11], [12], [13], Decision Tree [14], [15], [16], and Random Forest [17], [18], [19] have been applied to the detection and categorization of known malware. However, these approaches come with shortcomings in effectively managing the increasing variety and complexity of contemporary malware. Traditional machine learning algorithms often encounter trouble with feature engineering, depending largely on handcrafted features that may not be able to capture complex relationships to effectively counteract new threats. Traditional approaches are not flexible enough to keep up with the rapid changes in malware, which is continually evolving with sophisticated evasion techniques.

The traditional techniques of malware detection were largely superseded by the evolution of deep learning. Extracting the features of malicious code from raw malware samples and clustering these characteristics to obtain a malware classifier is what the deep learning models are based on, with the obvious benefits of high automation and minimal resource consumption [1]. The authors in [20], have introduced the *byte plot* method for malware visualization, where malware binaries are represented as pixels in an image, arguing that the visualization of the malware file into an image allows analysts to visually inspect the structure and patterns within files more intuitively and comprehensively compared to traditional text-based methods. Nataraj et al.[21] presented an approach for malware visualization and classification through image processing methods that convert malicious software binaries to grayscale images. They discovered visual similarities in layout and texture between malware images that belong to the same malware class. This visual similarity could be attributed to the popular practice of code reuse to develop new virus strains. By visualizing malware binaries as images, the visual features can be used to properly classify the malware samples without the need for disassembly or code execution. They revealed that even after the harmful program is loaded by the attackers for obfuscation, the packed malware still retains visual similarities. Researchers have adopted this technique to classify malware using CNN. However, CNN struggles with capturing positional relationships between features, due to the pooling layer, which reduces the size of the image by extracting the average or maximum value of a feature region and may lose some of the spatial hierarchy representation.

Sabour et al.[22] proposed Capsule Network (CapsNet) to address the CNN shortcomings. This kind of network uses the concept of "capsules" instead of neurons, which can save the relative position of features, the viewpoint, and other spatial information that can be crucial for image classification. Therefore, it optimizes the accuracy of image classification and recognition.

He et al. [23], proposed Residual Network (ResNet) to overcome the vanishing gradient problem caused by training very deep CNNs using skip connections among layers, which allows the network to learn residual functions rather than direct learning of entire mappings. Based on the above-mentioned networks, Alaoui-Elfels and Gadi [24] developed a deep learning network for image classification called ELU Residual Capsule Network (ER-Caps). It has shown its potential for classifying complex images. In this work, we adapt ER-Caps for malware detection and classification using malware images as input. We evaluate its performance in the different benchmark datasets with different data distributions to analyze the impact of class imbalance on model performance. We compare its performance with the current deep learning networks. We borrow concepts and methods from malware image visualization technology to generate images directly from unprocessed malware executable files [21], [25]. Such images facilitate malware analysis by extracting visual features. We first resize and normalize the malware images. Next, we process the inputs by ER-Caps to extract complex features, then identify whether the input is benign or malicious software for the detection task, and classify the malware into the appropriate family for the classification task. ER-Caps shows a high performance in malware detection and classification. The experiments were conducted on the Malimg dataset [21], MaleVis dataset [26], Blended dataset [27], VIRUS-MNIST dataset [79], and MMCC dataset [28] for malware classification, whereas for malware detection, we create a dataset for malware detection by combining benign executable files from [29] and [30] and malware files of Malimg dataset. The main contributions of this work include:

- We adapt ER-Caps to malware detection and classification and study its performance in benchmark malware datasets.
- We analyze the performance of ER-Caps on the Malimg, MaleVis, and Blended datasets to gain insights into the impact of class imbalance on the classification of malware images.
- We created a malware detection dataset that includes numerous benign samples and malware from 25 families, providing a diverse training set that reflects real-world environments.
- We compare the performance of ER-Caps with the baseline CapsNet [22] and several CNN-based models on malware detection and classification that illustrate its effectiveness in malware analysis. ER-Caps showed its high performance on image-based malware classification.

The rest of the paper is organized as follows: In Section 2, we review the related work in malware analysis. In Section 3, we describe our approach to malicious software detection and classification based on ER-Caps. In Section 4, we detail our experiments and discuss the results. Finally, in Section 5, we conclude and present future work suggestions.

II. Related Work

Caviglione et al. [31] reviewed in detail the evolution of malware analysis approaches. From the literature, we identify prior works from two perspectives, illustrated in Figure 2: traditional approaches and machine learning approaches. The traditional approaches include static-based, dynamic, and hybrid methods. The machine learning approaches are categorized into classical machine learning algorithms and deep learning algorithms.

A. Traditional approaches

1) Static analysis: static analysis was first proposed to determine whether the program is malware or not by identifying its structure, start-up mode, possible execution path,

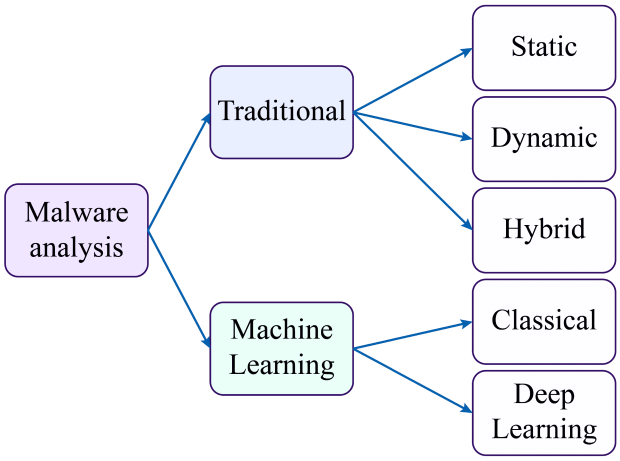

Fig. 2. Malware Analysis Taxonomy.

malicious behavior, and other binary code features [31], [32]. This approach analyzes the Portable Executable Files (PE files); first, it examines the headers of the PE files to validate their format and collect basic information such as the file's structure, and signature, including sections, imports, exports, and an optional header. Then, it decompiles the executable file using a disassembler tool such as OllyDbg and IDA Pro to obtain assembly language instructions. This provides details about the malware and enables an in-depth examination by scrutinizing: Application Programming Interface calls (API calls) [33], [34], [35], Control Flow Graph (CFG) [36], byte sequence n-grams [37], string signatures [38], and operation codes (opcodes) [39], [40] to identify anomalies and suspicious behavior that might point to malicious intent. Obfuscation techniques are constantly evolving and becoming more complex. They can circumvent static analysis approaches [41] by obscuring the actual functionality of the code and making it more difficult for analysis tools to spot particular elements or patterns in the code.

2) Dynamic analysis: dynamic analysis is carried out when static analysis has run out of options because of either packing or obfuscation. The dynamic analysis approaches allow monitoring the behavior and the actual actions executed by the executable program in a virtual environment [35]. Function call monitoring is a common malware dynamic analysis approach that involves analyzing the behavior of the program by observing and recording the sequence of functions that are invoked during the execution of code. The information flow tracking technique examines how an executable processes data and the data propagation mechanisms through the system [42]. While the function parameter analysis method entails tracking the parameter values and function return values [43], although polymorphic malware and code obfuscation strategies fail at dynamic analysis, each executable program needs to be executed in a virtual environment for a specific time to monitor its behavior, which requires high computational resources and is a very time-consuming process. Furthermore, the real runtime environment could differ significantly from the virtual environment, and malware may act differently in different environments. In certain conditions, malware actions may not be triggered and hence not logged [44].

3) Hybrid analysis: hybrid analysis gains the benefits of static and dynamic analyses, collecting information about malware through both analyses to increase the ability to detect malware correctly [45], [46]. Shijo and Salim [47] developed an integrated technique for detecting and classifying unfamiliar files. They extracted printable string information patterns using static analysis, and they extracted API calls using dynamic analysis. Islam et al. [48] used static analysis to extract function length, frequency, and printable string information features and used dynamic analysis to identify API calls and parameters. Ma et al. [49] proposed an approach that blended static and dynamic classifiers into one model to decrease false-positives in malware classification. Santos et al. [50] combined operations, system calls, exceptions acquired during dynamic analysis, and opcode frequency acquired during static analysis to develop a tool to detect unknown malware files. Damodaran et al. [6] carried out a comparative study of static, dynamic, and hybrid analysis techniques based on opcode and API call sequences using hidden Markov models. The findings indicate that a straightforward hybrid analysis is unlikely to be more effective than a fully static or fully dynamic analysis; it did not show consistent improvement.

B. Machine learning approaches

1) Classical: in order to deal with the issue of malware detection and classification, machine learning algorithms suggested a different approach [51]. They generally consist of four steps, starting with data construction, followed by feature engineering, then training and evaluating the model. The model can use static features, dynamic features, or both to achieve higher performance [50]. Many classical machine learning models were used for training, including logistic regression [52], k-Nearest Neighbor (kNN) [53], Support Vector Machine (SVM) [48], [54], [55], [56], Decision Tree [57], and Random Forest [58], [59]. Besides static features or dynamic features, image-based representation features have been proposed for malware analysis. The malware file is converted into an image, which preserves the malicious features. Therefore, benign samples can be distinct from malicious samples. Conti et al. [20] developed Byteplot, a visualization approach for displaying binary data items as images, proving that visual examination of binary data allows for the separation of structurally diverse data fields. Nataraj et al. [21] suggested that image structure analysis is more suitable for malware classification as opposed to other malware analysis approaches. They convert the content of the malware file to a binary string of zeros and ones. Then, they reshaped the binary vector into a matrix, and therefore the malicious software file is presented as a grayscale image. This technique classifies malicious software based on the image texture, which can withstand obfuscation methods. Malware from the same family represents a similar texture, which can be easily detected using malware images. Using Gabor filters, the authors collected features from malware images; afterward, they used the kNN for classification, where they achieved a classification accuracy of 98% for the dataset consisting of 25 malware classes and 9339 images. Later, Kancherla et al. [54], [55] converted the PE malware file into the image and extracted malicious features to feed

SVM and kNN. They reached an accuracy of 95% on their malware detection dataset. However, classical machine learning algorithms heavily rely on feature engineering as well as complicated or expert features to fulfill the learning process and classify malware with high performance. The final result is tied to the selected features, which can be highly subjective since it relies on the expertise and insight of the experts. These algorithms necessitate manual effort and cannot be conducive to efficient real-time malicious software analysis.

2) Deep learning: CNNs have had impactful success in computer vision [60]. CNNs, as a deep learning algorithm, have the ability to classify images directly from raw pixel values without using complex feature engineering approaches. Many researchers used different CNN models to categorize malware images, using the visualization of malware files as images [3], [44], [51], [61]. Kiger et al. [62] proved the viability of classifying malware represented as grayscale images by evaluating and analyzing the performance of several CNN architectures in classifying malware images. Jayasudha et al. [27] studied the performance of XceptionNet, ResNet50, EfficientNetB0, DenseNet169, InceptionResNetV2, and VGG16 on imbalanced datasets. They observed that XceptionNet and VGG16 were sensitive to imbalanced data, while ResNet50, DenseNet169, and EfficientNetB0 could handle imbalanced data. Chaganti et al. [63] proposed the EfficientNetB1 network to perform malware classification using a pre-trained CNN.

Rezende et al. [64] used pre-trained VGG16 layers for bottleneck feature extraction and then used these features to train an SVM to classify malware. In another work, these authors presented a deep learning approach based on the pre-trained ResNet50 to classify the Malimg dataset [65]. Li et al. [66] combined ResNet50 with a self-attention mechanism and data augmentation to detect malware. Maryam et al. [67] merged the extracted features from pre-trained Inception-v3 and AlexNet with features extracted from malware images using segmentation-based fractal texture analysis to classify the Malimg dataset.

Other works used ensemble models to boost malware classification. Vasan et al. [61] proposed an ensemble CNN that contains VGG16, ResNet50, and SVM to detect packed and unpacked malware. Roberts et al. [68] introduced an ensemble learning-based methodology for malware detection. They stacked an ensemble of dense neural networks and CNN and used the Extra Trees algorithm as a meta-learner to perform classification.

Despite the significant results of the above-mentioned works, any CNN requires a substantial amount of training data that is not easily accessible in the realm of malware detection. Moreover, the pooling layer that most CNN architectures rely on may cause feature position details to be lost, which may be critical to malware classification [69]. Attempting to overcome the shortcomings of CNN, Sabour et al. [22] proposed the capsule network, which uses capsules to represent low-level and high-level features. These capsules are related through the dynamic routing algorithm. This network is considered an important deep-learning model for the next generation. It demonstrated its high ability for feature extraction compared with the classical neural network, and it showed its potential in various fields [70], [71], and [72].

some researchers proposed to use the Capsule Network for the cybersecurity domain. Zhang et al. [73] applied CapsNet for the malware classification task, compared its performance with CNN, and demonstrated its effectiveness in malware detection. Wang et al. [74] introduced an analysis method based on malware color image visualization techniques and CapsNet. Çayır et al. [75] proposed a random CapsNet for imbalanced malware using bootstrap aggregating methods. Wang et al. [76] introduced an approach to extracting the operation code information by combining static and dynamic analysis, then constructing and visualizing the n-gram sequence, generating malware images, and lastly employing CapsNet for classification. Zou et al. [77] built a malware classification model that used dynamic features and CapsNet to classify malware. In this paper, we discusse the effectiveness of ER-Caps for malware analysis and the role and relevance of this network.

III. Methodology

A. Malware datasets

In this work, we used five publicly available malware datasets to evaluate the ER-Caps model for the malware classification task:

- The Malware Images (Malimg) dataset [21] includes 9342 malware PNG images categorized into 25 distinct malware classes (Table 1). It is a highly imbalanced dataset, as it is displayed in Figure 3. The images have been created from PE files, which are malware binary files. The PE files were transformed into an 8-bit vectors, and then the resulting array was organized into a 2D array where each byte was interpreted as one pixel to create grayscale images having different sizes and a color depth of three (Figure 4).
- The Malware Evaluation with Vision (MaleVis) dataset [78] includes 14226 RGB images belonging to 26 malware families, which consist of 25 malware and 1 benign, as shown in Table II. For the malware classification, we use only the images of the 25 malware classes. The distribution of MaleVis samples among the different malware classes is shown in Figure 5, and it is observed that the dataset is perfectly balanced.
- The Blended dataset [27] was created by blending five principal classes of the Malimg dataset with all MaleVis dataset classes to obtain a new dataset exhibiting a moderate class imbalance compared to the Malimg and MaleVis datasets (Figure 6).
- The Microsoft Malware Classification Challenge (MMCC) dataset [28] consists of 21741 samples, representing 9 malware classes. Each sample has a corresponding PE file and assembly file, which store metadata information like function calls, strings, and memory allocation. Table III and Figure 7 display the distribution of various MMCC classes present in the training dataset.
- VIRUS-MNIST dataset [79] consists of 51880 malware image samples from nine malware families and one benign software. Malware images are generated through formatting the first 1024 bytes into a 32x32 resolution image. Table IV and Figure 8 display each malware class distribution in VIRUS-MNIST dataset.

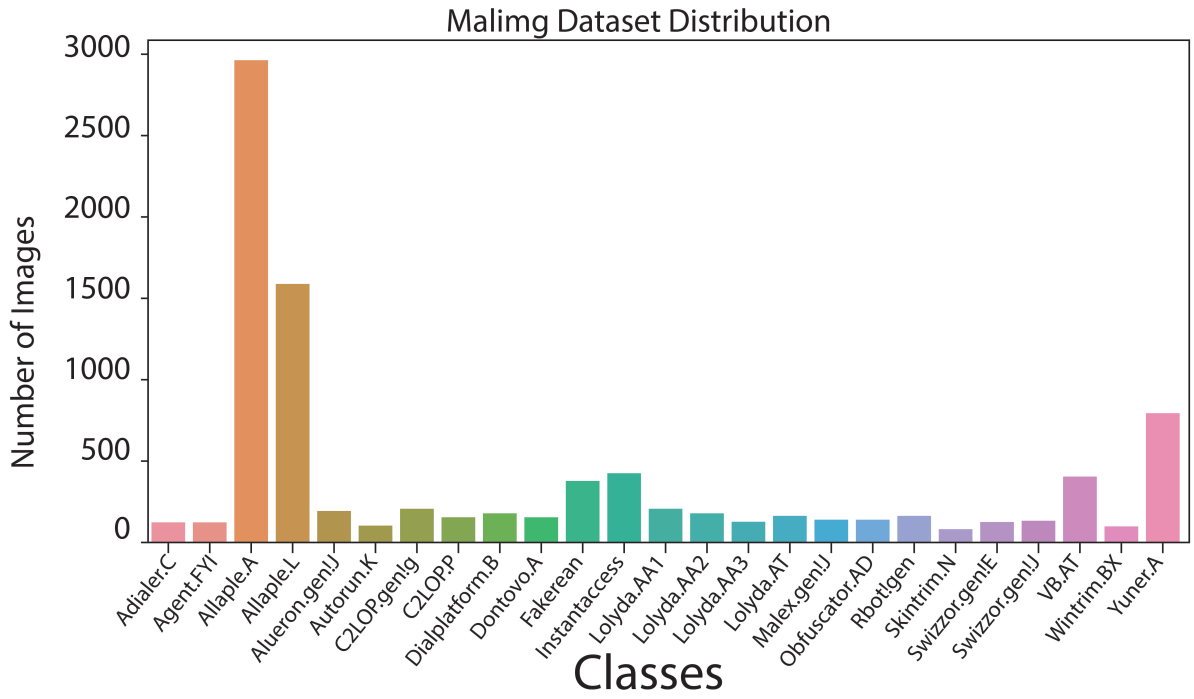

Fig. 3. Malimg data distribution.

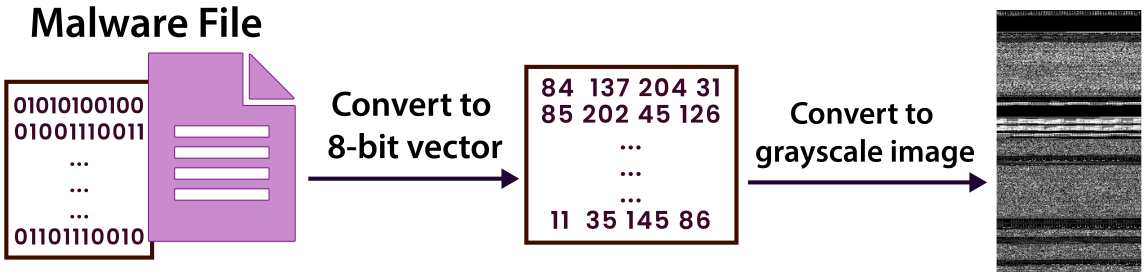

Fig. 4. Visualizing malware file as an image.

For the malware detection task, we created a dataset of 27371 samples divided into two classes: malware and safe software (benign). For the malware class, we aggregated all the malware images of the Malimg datasets into a malware class. For the benign class, we combine 1000 non-malware executable files from [29] and 14355 non-malware executable files from [30] that were collected from various resources. We transform each benign file to a grayscale image using the method of Nataraj et al [21], where the image dimensions are determined based on the file size. The binary data from the executable files are converted into a vector of unsigned 8-bit integers, each representing a pixel value. The vector is reshaped into a 2D matrix, then converted into a PNG image. Our dataset contains many benign samples, making it close to practical application in a real environment.

B. Sample Visualization and pre-processing

The Malimg, the MaleVis, the Blended, and the VIRUS-MNIST datasets consist of malware images. For the MMCC dataset, we transform the hexadecimal data of the PE files into 2D arrays. Every pair of hexadecimal format represents one byte, which represents values between 0 and 255 in decimal. Therefore, in a 2D array, each byte is interpreted as a grayscale pixel intensity value.

Image pre-processing is a preparatory step that fixes image resolution and normalizes data before feeding it into a training model. The MaleVis and the VIRUS-MNIST datasets contain RGB images, while the Malimg and MMCC datasets contain grayscale images. Consequently, the Blended dataset and the malware detection dataset comprise both grayscale and RGB images. Furthermore, the images of the MMCC and the Malimg datasets have different sizes and must be transformed into a single image size. We converted the images of all datasets to RGB and resized them to 32 by 32 (Figure 9).

C. Overview of Architectures used

Figure 10 displays the data flow architecture diagram for malware detection and classification for different malware datasets. All the images in the datasets are resized, normalized, and split into a train set and a test set.

Following this, the data is fed to the ER-Caps (Figure 11), the details of the network architecture are shown in Table V. ER-Caps process the input image using a convolutional layer followed by a set of three residual blocks to extract the

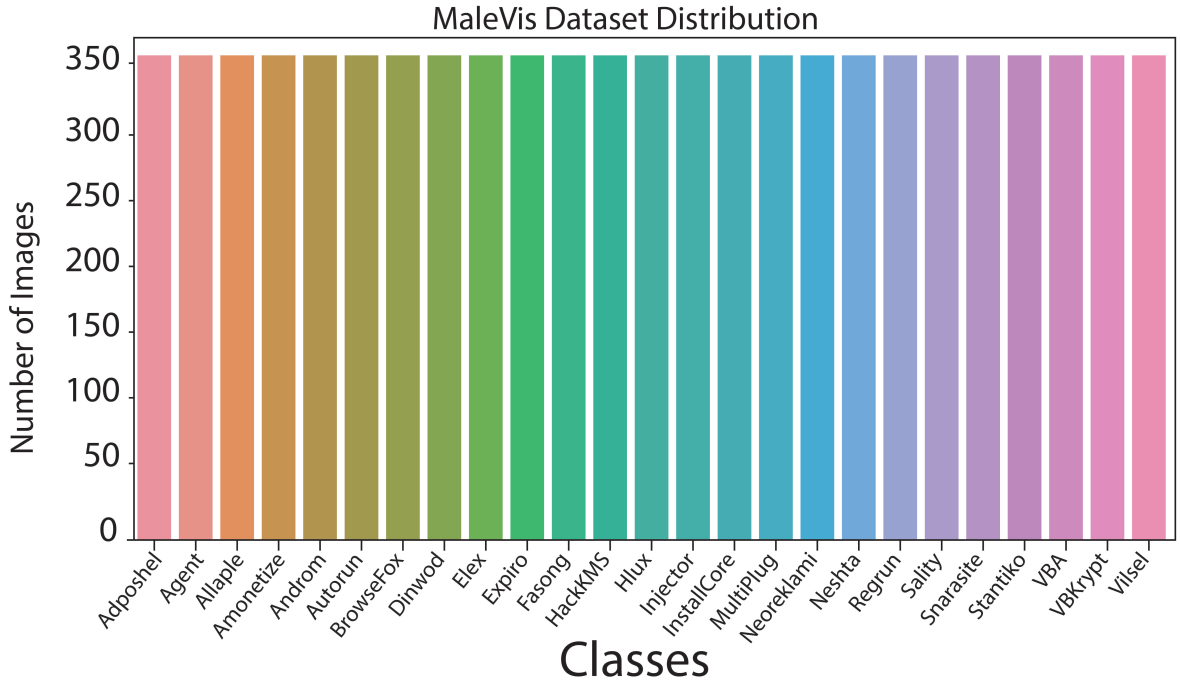

Fig. 5. MaleVis data distribution.

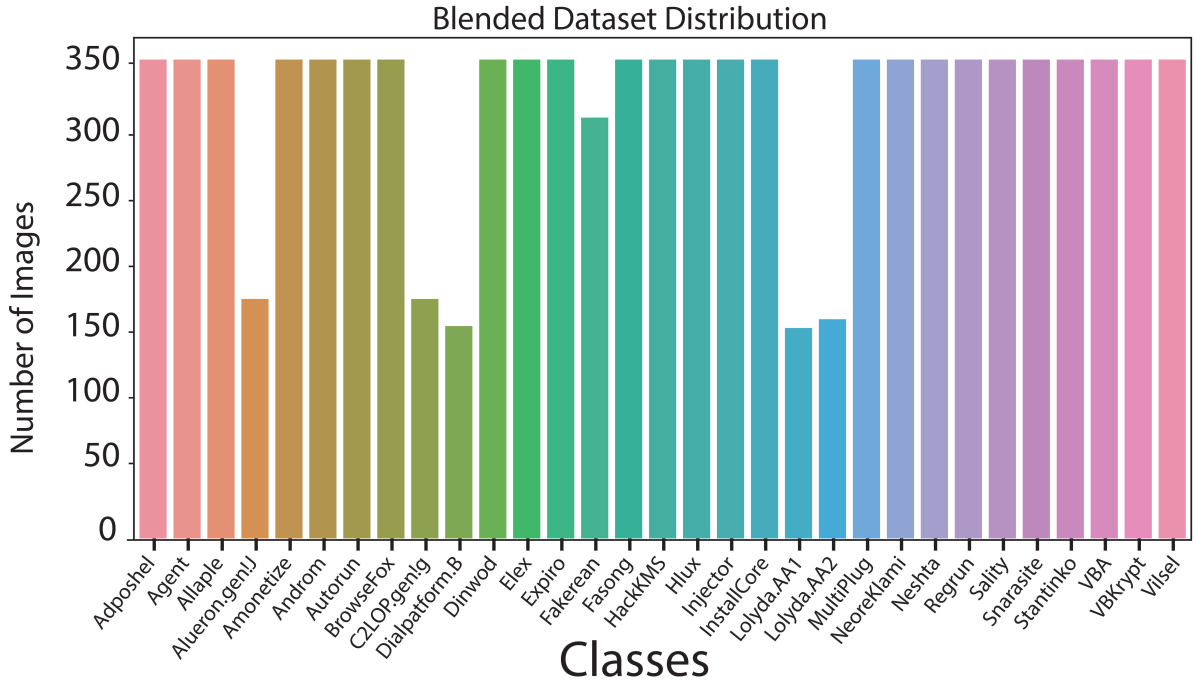

Fig. 6. Blended dataset distribution.

malware image features. Each residual block processes the input through two convolutional layers (Conv1 and Conv2) and then adds or multiplies the input back to the output via a shortcut connection. This connection enables the network to learn the residual between the input and output, making it easier to train and improve gradient flow through it. The first and second residual blocks use the addition operator to inject the residual output of the previous layer or block into their output, while the third residual block uses element-wise multiplication to combine the residual identity with its output. The element-wise gate shows its potential to improve the model's performance and select meaningful features from complex images [24], [80], [81].

The Conv1 and Conv2 are activated, respectively, using the RELU and the ELU activation units. RELU guarantees a lower run time, and the ELU mitigates the vanishing gradient problem. The convolutional layers are created using 256 kernels of 3 by 3 size and a stride of 1. The Batch Normalization (BN) was performed after the convolutional layers to stabilize and accelerate the training process by

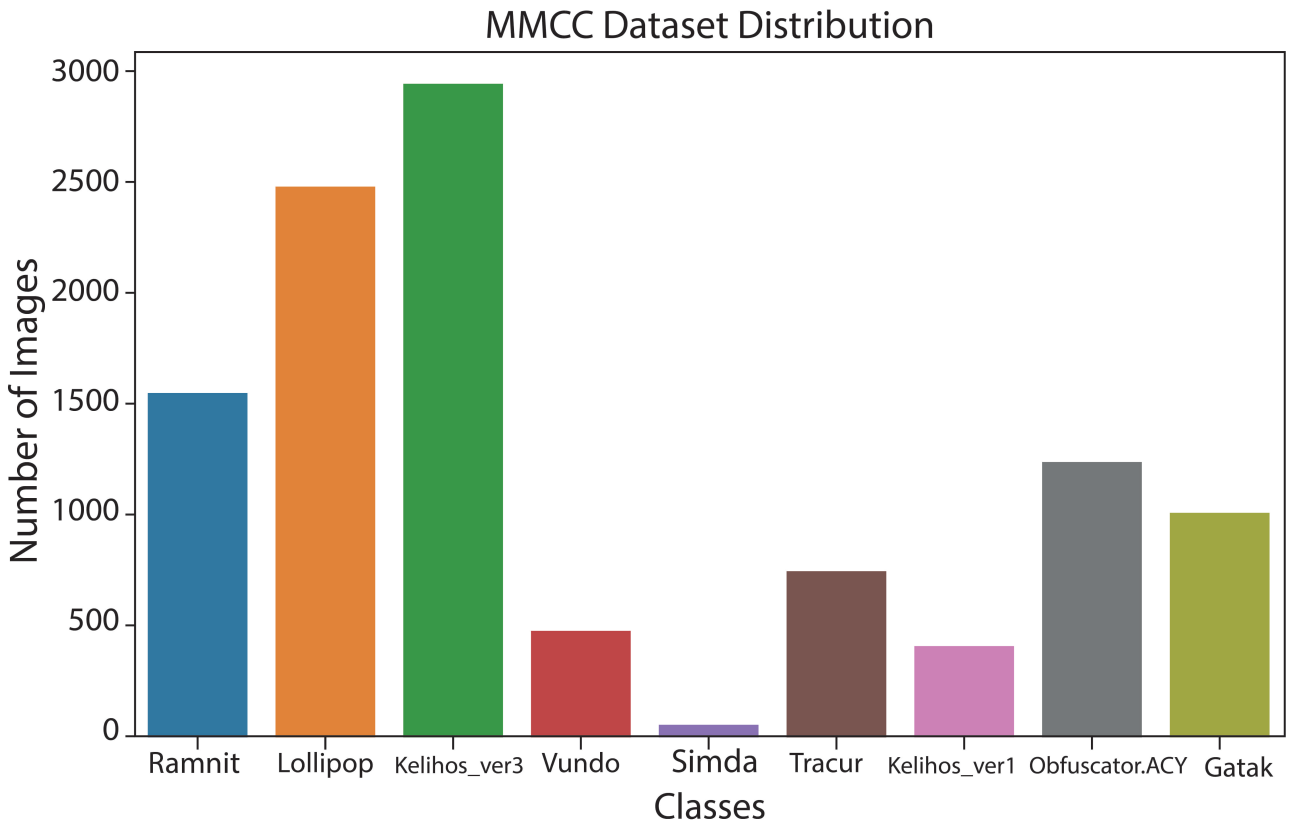

Fig. 7. MMCC data distribution.

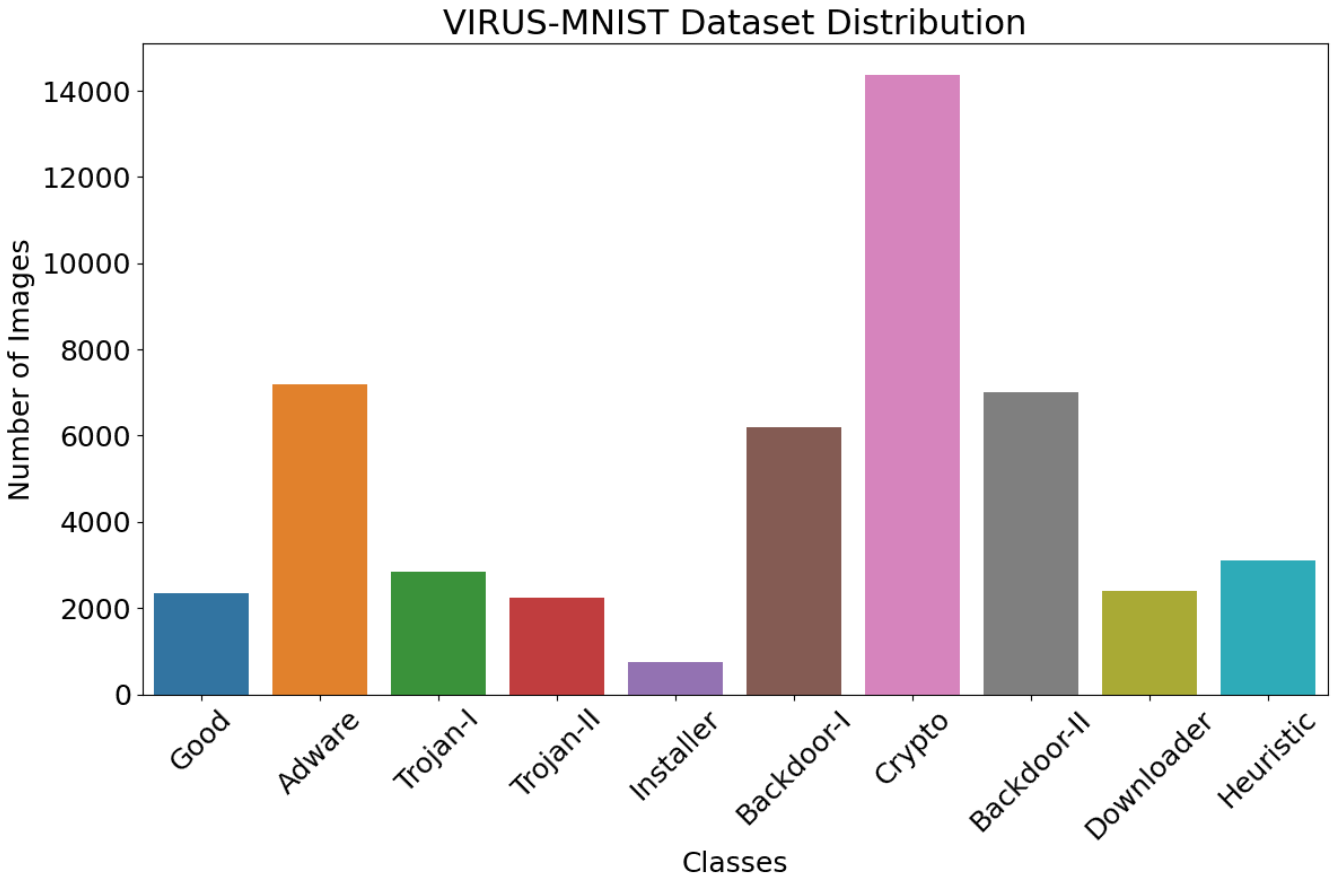

Fig. 8. VIRUS-MNIST data distribution.

Fig. 9. Malware samples from the Malimg dataset after pre-processing.

smoothing variations of input distribution in the network. BN acts as a regularizer by introducing slight noise during training, as it uses the statistics of mini-batches, which vary slightly from batch to batch, helping to bring down overfitting and benefit model generalization. A dropout layer of 5% rate is used after the residual blocks to help the model generalize better to unseen data.

The resulting feature maps of the residual subnet are then mapped to primary capsules in the PrimCaps layer. This capsule layer applies small convolutional kernels of 4x4 size and 32 channels, then encapsulates the output feature maps into vectors of 8 scalars called primary capsules. These capsules serve to detect and represent low-level patterns or features within an input image. Then, the primary capsules pass their activation information to class capsules in the ClassCaps layer through the Dynamic Routing (DR) algorithm to determine whether the input is malicious or benign in case of malware detection and to determine the malware class in case of malware classification. The class capsules encode high-level representations of object classes into vectors of length 16; they can detect the presence or absence of their respective classes within the input.

The DR algorithm routes low-level features represented by the primary capsules to high-level features represented by the class capsules based on the likelihood of the presence of a particular object within the input. The general DR process is as follows:

1) $u_i \in R^{k\times 1}$, $i = 1, 2, ..., n$, represents the output of the primary capsule, where n is the number of capsules and k is the vector length (the number of neurons within each capsule). This output is multiplied by the transformation matrix $W_{ij} \in R^{p\times k}$ to compute the prediction vector $\hat{u}_{j|i}$. p represents the length of the class capsule, and j represents a class capsule. W_{ij} provides significant spatial information between different entities, and it is learned during the back-propagation algorithm.
2) Calculating the weighted sum of all the resulted prediction vectors:

$$S_j = \sum_i C_{j|i}\hat{u}_{j|i} \tag{1}$$

$C_{j|i}$ is called the coupling coefficient; the $C_{j|i}$ for a parent capsule j is equal to 1. It represents the likelihood distribution of capsule i to activate capsule j.
3) Applying a squash function to the S_j to calculate the V_j that represents the probability of existence, which takes values between 0 and 1.

$$V_j = \frac{\|S_j\|^2}{1+\|S_j\|^2}\frac{S_j}{\|S_j\|} \tag{2}$$

The coupling coefficient is refined iteratively as follows:

$$C_{j|i} = \frac{e^{b_{ij}}}{\sum_k e^{b_{ik}}} \tag{3}$$

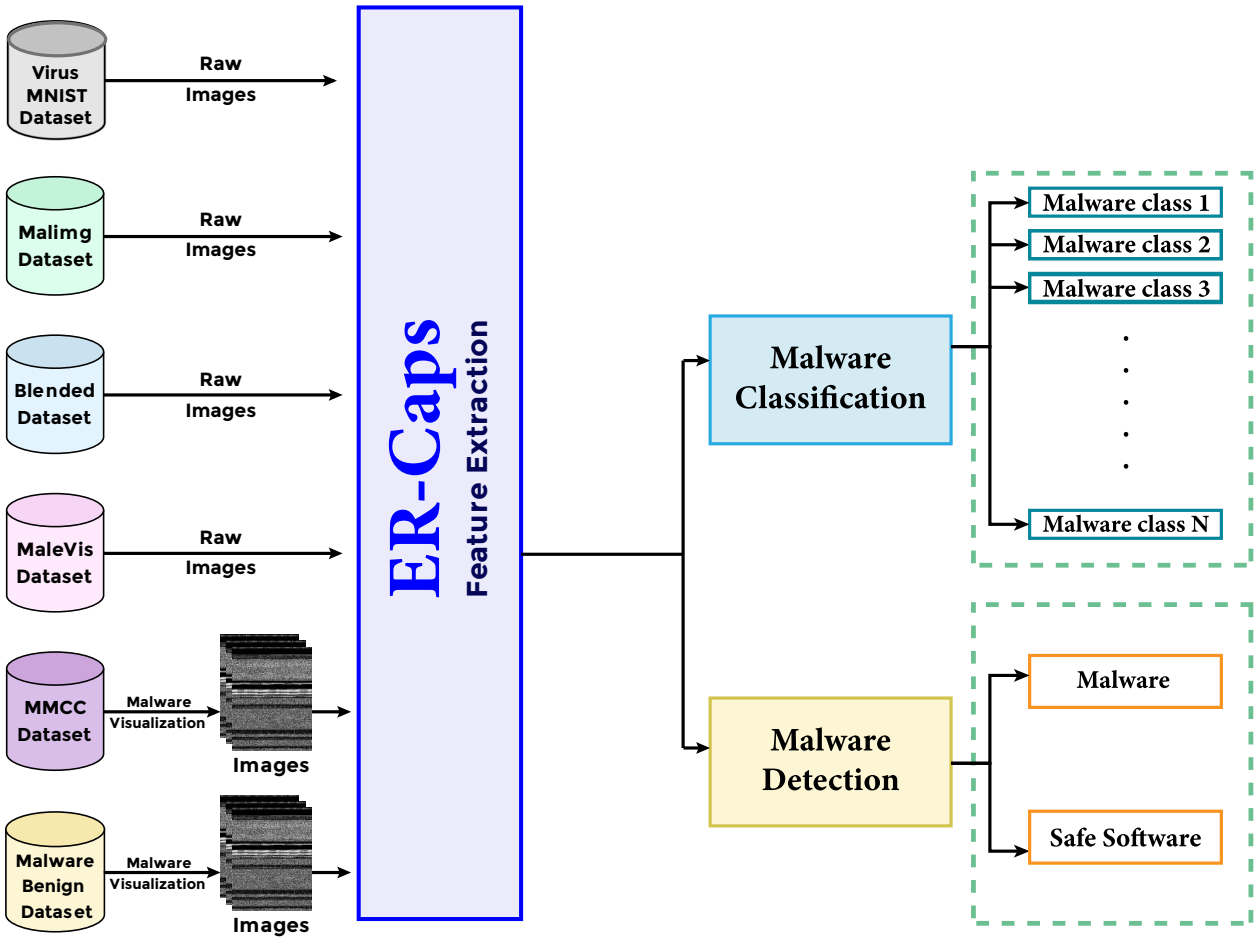

Fig. 10. Architecture Diagram.

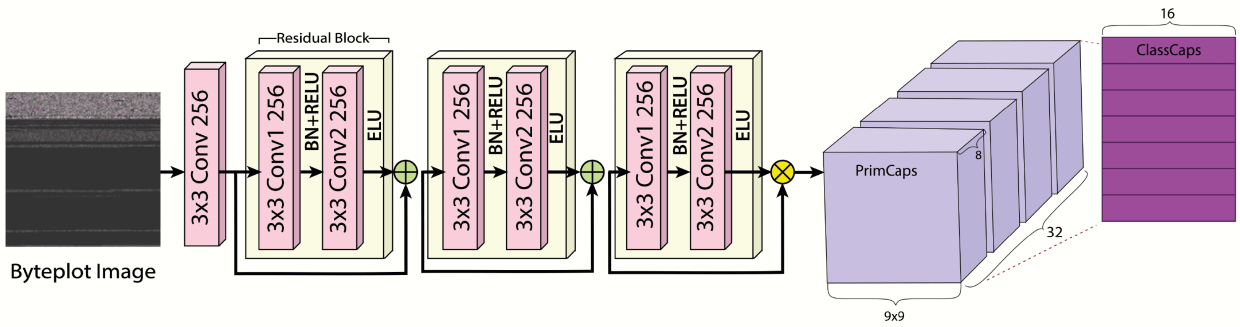

Fig. 11. ER-Caps for malware analysis.

Where b_{ij} represents the matching degree between the class capsules and the primary capsules.

$$b_{ij} = b_{ij} + \hat{u}_{j|i} V_j \tag{4}$$

This DR mechanism enables the network to iteratively update its predictions, hence improving its capacity to localize and recognize objects.

The ClassCaps layer is connected to a decoder subnet to reconstruct the input data, which consists of three layers of neurons that are fully connected. Where the last layer is reshaped into a matrix to represent the reconstructed image. In this case, the loss is calculated by summing the classification loss and the reconstruction loss, as described in [82].

In this work, we compare the performance of the ER-caps with baseline CapsNet [82] in malware analysis. This network has a shallow architecture, which treats the input data as a convolutional layer (256 filters, 9x9 size, and stride of 1, activated by the RELU function). Followed by the PrimCaps layer with eight convolution operations of 32 channels, 9x9 kernels, and a stride of 2. Next, the results are routed to the ClassCapss layer using the DR algorithm to carry out the classification.

TABLE I
MALIMG CLASSES AND THEIR RESPECTIVE NUMBER OF INSTANCES.

Class Name	Samples Number	Malware Category
Adailer.C	122	Dialer
Allaple.A	2949	Worm
Agent.FYI	116	Backdoor
Allaple.L	1591	Worm
Autorun.K	106	Worm:AutoIT
Alueron.gen!J	198	Worm
C2LOP.P	146	Trojan
C2LOP.gen!g	200	Trojan
Dontovo.A	162	Trojan Downloader
Dialplatform.B	177	Dialer
Fakerean	381	Rogue
Lolyda.AA1	213	Password Stealer
Instantaccess	431	Dialer
Lolyda.AA2	184	Password Stealer
Lolyda.AT	159	Password Stealer
Lolyda.AA3	123	Password Stealer
Malex.gen!J	136	Trojan
Rbot!gen	158	Backdoor
Obfuscator.AD	142	Trojan Downloader
Skintrim.N	80	Trojan
Swizzor.gen!I	132	Trojan Downloader
Swizzor.gen!E	128	Trojan Downloader
VB.AT	408	Worm
Yuner.A	800	Worm
Wintrim.BX	97	Trojan Downloader

D. Evaluation Metrics

According to the literature review, the majority of articles utilize an accuracy metric for an imbalanced multiclass dataset, which is inappropriate because it is biased toward the majority classes. Furthermore, for malware detection, false positives are more expensive than false negatives, so the weighted precision metric is the most appropriate metric for this task because it evaluates the model's ability to correctly identify the minority class, and can be supported by weighted recall and weighted F1-score, as shown in the formulae below:

$$\text{Weighted Precision} = \frac{\sum_i \text{Precision}_{\text{Class}_i} \times \text{Support}_{\text{Class}_i}}{\text{Total\_Samples}} \quad (5)$$

$$\text{Weighted Recall} = \frac{\sum_i \text{Recall}_{\text{Class}_i} \times \text{Support}_{\text{Class}_i}}{\text{Total\_Samples}} \quad (6)$$

$$\text{Weighted F1-score} = \frac{\sum_i \text{F1-score}_{\text{Class}_i} \times \text{Support}_{\text{Class}_i}}{\text{Total\_Samples}} \quad (7)$$

Knowing that:

$$Precision = \frac{TP}{TP + FP} \quad (8)$$

$$Recall = \frac{TP}{TP + FN} \quad (9)$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (10)$$

Where:

- TP: True positives;
- TN: True negatives ;
- FP: False positives;
- FN: False negatives.

TABLE II
MALEVIS CLASSES AND THEIR RESPECTIVE NUMBER OF TRAINING INSTANCES.

Class Name	Samples Number	Malware Category
Adposhel	350	Adware
Agent	350	Trojan
Allaple	350	Worm
Amonetize	350	Adware
Androm	350	Backdoor
Autorun	350	Worm
BrowseFox	350	Adware
Dinwod	350	Trojan
Elex	350	Trojan
Expiro	350	Virus
Fasong	350	Trojan
HackKMS	350	Riskware
Hlux	350	Worm
Injector	350	Trojan
InstallCore	350	Adware
MultiPlug	350	Adware
Neoreklami	350	Adware
Neshta	350	Virus
Benign	350	-
Regrun	350	Trojan
Sality	350	Virus
Snarasite	350	Trojan
Stantinko	350	Trojan
VBA	350	Macro Malwares
VBKrypt	350	Trojan
Vilsel	350	Trojan

TABLE III
MMCC CLASSES AND THEIR RESPECTIVE NUMBER OF TRAINING INSTANCES.

Class Name	Samples Number	Malware Category
Kelihos_verl	398	Backdoor
Kelihos_ver3	2942	Backdoor
Gatak	1013	Backdoor
Lollipop	2478	Adware
Ramnit	1541	Worm
Obfuscator.ACY	1228	Any kind of obfuscated malware
Simda	42	Backdoor
Vundo	475	Trojan
Tracur	751	TrojanDownloader

IV. EXPERIMENTS AND RESULTS

The models were trained on a single computer with 16 GB of RAM, NVIDIA GeForce RTX 3070 GPU, and an 8-core CPU. The software environment is Windows 11, Python 3, and Pytorch.

A. Malware classification

We downsample the Malimg images to a fixed size of 32x32x3 to train the network. In order to keep the model

TABLE IV
VIRUS-MNIST CLASSES AND THEIR RESPECTIVE NUMBER OF SAMPLES.

Class Name	Samples Number
Good	2516
Adware	7684
Trojan-I	3037
Trojan-II	2404
Installer	796
Backdoor-I	6662
Crypto	15377
Backdoor-II	7494
Downloader	2571
Heuristic	3339

TABLE V
ER-CAPS STRUCTURE.

Layers	Kernels	Strides	Padding
Conv+BN+RELU	[256, 3, 3]	[1, 1]	–
1st and 2nd residual block :			
Conv1+BN+RELU	[256, 3, 3]	[1, 1]	–
Conv2+BN+ELU	[256, 3, 3]	[1, 1]	–
Dropout	–	–	–
3rd residual block :			
Conv1+BN+RELU	[256, 3, 3]	[1, 1]	–
Conv2+BN+ELU	[256, 3, 3]	[1, 1]	[1, 1]
Dropout	–	–	–
PrimaryCaps :			
8 conv	[32, 4, 4]	[2, 2]	–
ClassCaps	–	–	–

simple and reduce the training parameters, we use the ER-caps network with 32 primary capsules and without the decoder part since we do not need malware reconstruction to identify or classify malware. Table VI supports our choice; it shows that using the decoder part and doubling the number of primary capsules makes the model more complex and does not improve its performance in malware classification.

1) The effect of class imbalance: we compare the performance of ER-Caps on the MaleVis dataset, Malimg dataset, and Blended dataset to evaluate the robustness of the model on data imbalances. Table VII displays the evaluation metrics of ER-Caps across three datasets. It has been observed that the model is capable of handling different data distributions. The values of precision and recall are more than 92% for all three datasets, indicating that the model is accurate and generalizes well. Moreover, the results suggest that the ER-Caps performs well on a highly imbalanced dataset (Malimg); it achieved 98% precision and recall; and it is capable of minimizing FP (precision) and capturing the majority of positive occurrences (recall).

2) Resistance to obfuscation: to evade anti-killing techniques by security tools, malicious software incorporates a polymorphic engine into their malicious code, generating code variants with different byte sequences but maintaining the same functionality. In our experiments, the Obfuscator.AD and Obfuscator.ACY classes, as well as the *worm* category (in which the code section is polymorphically encrypted using dynamic encryption keys), represent this kind of obfuscation. Nonetheless, our proposed classifier is capable of classifying these samples with high precision, achieving a precision of over 99% (see Figure 14). The image textures applied for classification supply more feature resistance concerning obfuscation techniques.

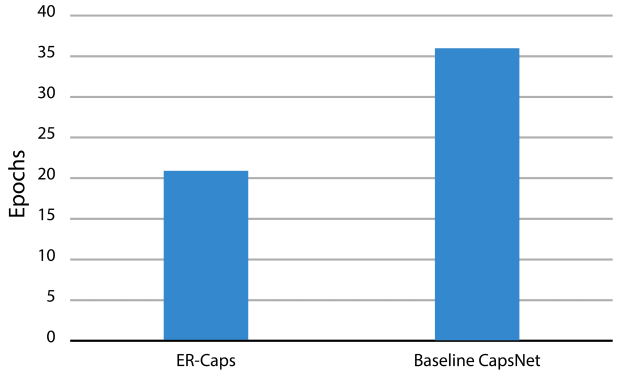

Fig. 12. Convergence epochs.

3) ER-caps Vs CapsNet: for performance comparison, we first compare the performance of the ER-caps with the baseline CapsNet in the malware classification images on the imbalanced dataset: Malimg and MMCC. As the focus of this paper is malware detection and classification, we only consider the classification part of the baseline CapsNet without the decoder part. Table VIII shows the F1-score for each model. This metric balances precision and recall, taking into account both false positives and false negatives. According to the experimental results, ER-Caps outperforms the baseline CapsNet by 15% and 5.6%, respectively, on the Malimg and MMCC datasets in terms of the F1-score. Moreover, from the convergence plot represented by Figure 12, it is observed that ER-Caps requires a smaller number of epochs for convergence, 21 epochs compared with 36 epochs for the baseline CapsNet.

Figure 13 displays the confusion matrix of baseline CapsNet classification on the Malimg dataset. There is a lot of confusion between C2LOP.P and C2LOP.gen!g. Similar trends can be observed between Swizzor.gen!E and Swizzor.gen!I classes, as well as between Allaple.A and Allaple.L classes and between Allaple.A and Malex.gen!J classes, besides total confusion between Autorun.k and Yuner.A classes. The model classes all Autorun.k samples as Yuner.A classes, and it is incapable of making a distinction between these two classes. Figure 14 presents the confusion matrix of the ER-Caps. This model alleviates the class-confusion issue; it shows a very low confusion rate on the classification of the Malimg dataset, and it is capable of classifying most malware classes correctly with a precision of more than 97% for 20 classes compared with 11 classes for the baseline CapsNet.

4) Generalization analysis: throughout the training process of the Malimg dataset, we monitored test loss. As shown in Figure 15, the test loss consistently decreased over epochs. This trend indicates that the model generalizes well, learning patterns from the training data without overfitting. We also tracked precision metrics (Figure 16), the training precision reached 99%, while the test precision stabilized at 98% after convergence. The small gap of approximately 1% between training and validation precision suggests that the model

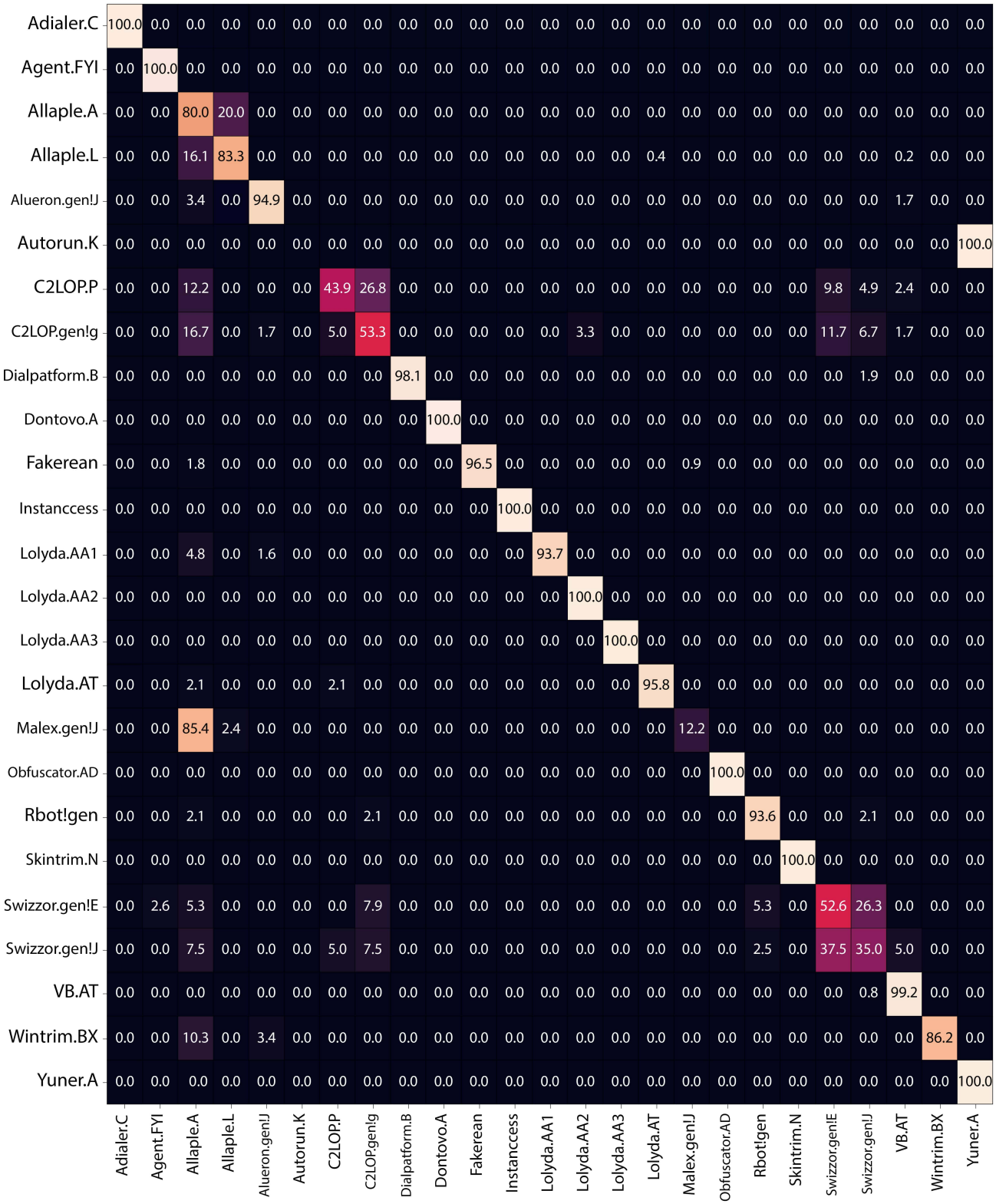

Fig. 13. Confusion matrix of baseline CapsNet classification results on Malimg dataset.

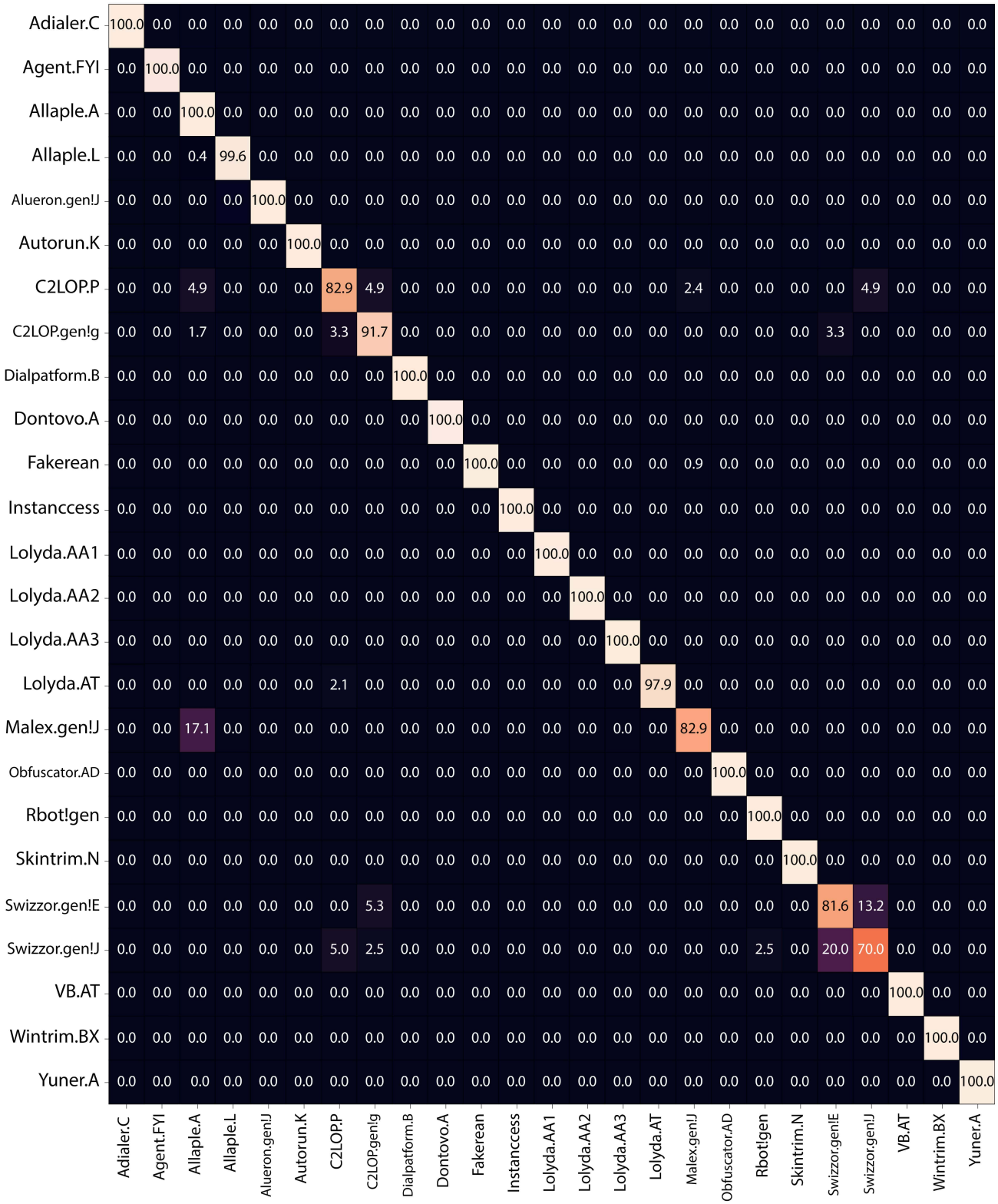

Fig. 14. Confusion matrix of ER-Caps classification results on Malimg dataset.

TABLE VI
COMPARISON OF ER-CAPS ARCHITECTURES.

Decoder	No. primary capsules	Parameters (M)	Precision %	Recall %	F1-score%
yes	64	9.4	95.54	95.36	95.30
yes	32	8.4	95.57	95.48	95.25
No	32	4.5	98.62	98.61	98.43

TABLE VII
EVALUATION METRICS FOR ER-CAPS IN DIFFERENT MALWARE DATASETS.

Dataset	Precision %	Recall %
Malimg	98.62	98.61
MaleVis	93.46	92.82
Blended	93.69	93.38

TABLE VIII
F1-SCORE OF THE ER-CAPS AND THE BASELINE CAPSNET ON THE MALIMG AND THE MMCC DATASETS.

	Malimg	MMCC
ER-Caps	98,61%	94.20%
Baseline CapsNet	83.36%	88.40%

TABLE IX
CROSS VALIDATION.

Fold 1	98.88%
Fold 2	98.84%
Fold 3	98.7%
Fold 4	98.73%
Fold 5	97.7%
Average Precision	98.61%
Precision Variance	0,13
Precision Standard Deviation	0,36

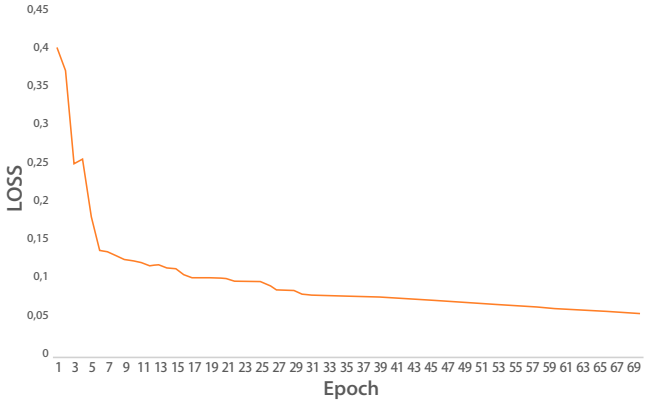

Fig. 15. Test loss.

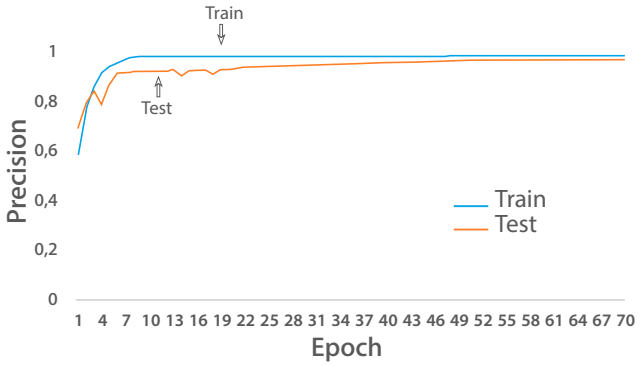

Fig. 16. Learning curves.

TABLE X
COMPARISONS OF ER-CAPS WITH THE EXISTING DEEP LEARNING NETWORKS ON THE MMCC DATASET.

Model	Enhanced Training Strategies	F1-score %
DenseNet201 [63]	Transfer Learning	94.0
InceptionV3 [63]	Transfer Learning	88.0
ResNet50 [63]	Transfer Learning	83.0
VGG16 [63]	Transfer Learning	95.0
CNN [83]	Data Augmentation	94.5
MalCaps [73]	Data Augmentation + Class Weight	94.43
CNN [44]	–	94.0
FACILE [84]	–	92.62
Baseline CapsNet	–	88.40
ER-Caps	–	94.02

generalizes well to the test data.

To further assess generalization, we employed 5-fold cross-validation. By splitting the Malimg dataset into 5 subsets, we trained and validated the model 5 times, each time using a different subset as the test set while the remaining 4 subsets were used for training. As shown in Table IX, the cross-validation results consistently showed similar performance across different folds. The average precision across folds is 98.61%, with a low variance and standard deviation across the 5 folds, indicating that the model's performance is stable and consistent, reinforcing the robustness of the model's generalization capabilities. Moreover, we evaluated the model's ability to generalize to completely unseen data using a separate split of data that was not used during training. The model achieved a precision of 97.56% on the unseen data and 98.61% on the test set. This consistent performance across both the unseen data and test set further confirms that the model generalizes well, effectively applying learned patterns to new, unseen data.

5) Comparison of ER-Caps Results with Deep Learning Networks on Malware Classification: there are several deep learning architectures applied to malware classification. To further assess the effectiveness of ER-Caps, we benchmarked it against cutting-edge networks from the literature that have been tested on the Malimg, the MMCC, and the VIRUS-MNIST datasets. The results are shown in Table X, Table XI, and XII for MMCC, Malimg, and VIRUS-MNIS datasets respectively.

The models chosen are based on deep learning CNN algorithms that classify malware based on raw images as well as ER-Caps. For the MMCC dataset, it is observed that ER-

TABLE XI
COMPARISONS OF DIFFERENT NETWORKS ON THE MALIMG DATASET.

Model	Enhanced Training Strategies	F1-score %
XceptionNet [85]	Data Augmentation	85
EfficientNetB0 [85]	Data Augmentation	96
ResNet50 [85]	Data Augmentation	95
VGG16 [85]	Data Augmentation	79
DenseNet169 [85]	Data Augmentation	95
InceptionResNetV2 [85]	Data Augmentation	91
BAT [61]	Data Augmentation	94.3
IMCEC [61]	Transfer Learning + Ensemble Learning	99.4
CNN [62]	Class Weight	94.0
CNN [44]	-	94.04
FACILE [84]	-	97.05
Baseline CapsNet	-	83.36
ER-Caps	-	98.43

TABLE XII
COMPARISONS OF DIFFERENT NETWORKS ON THE VIRUS-MNIST DATASET.

Model	F1-score %
CapsNet [84]	84.65
MalCaps [84]	74.48
Efficient-CapsNet [84]	86.62
DA-CapsNet [84]	88.15
MLCN [84]	83.58
ResNet152 [84]	86.47
EfficientNet [84]	87.20
MCFT-CNN [84]	88.42
VGG16 [84]	88.04
FACILE [84]	88.07
ER-Caps	91.97

Caps achieved comparable results to other CNN models without any data augmentation, transfer learning, or ensemble learning techniques. Unlike EfficientNetB0, DenseNet169, ResNet50, VGG16, CNN [62], and CNN [44] that use max pooling or average pooling, and InceptionResNetV2, BAT, and IMCEC that use average pooling and max pooling to reduce the image size, ER-Caps do not use any pooling approach to preserve spatial relationships among features.

For the Malimg dataset, it was observed that ER-Caps achieved a high malware classification rate. IMCEC [61] exceeds ER-Caps by 1% in terms of F1-score; however, IMCEC needs more than 159.94 million trainable parameters since it is based on an ensemble of CNN architectures to get this performance. The IMCEC model is too complex compared with ER-Caps, which uses only 4.59 million trainable parameters to classify the Malimg dataset. Overall, ER-Caps outperforms most malware classification models that use enhanced training strategies to boost their performance and achieved a F1-score of 98.43% on classifying malware in the Malimg dataset.

For the VIRUS-MNIST dataset, the model demonstrated evident superiority over all capsule network models and CNN-based models, achieving an F1-score of 91.97%, which confirms the ER-Caps capability in accurately classifying highly imbalanced malware dataset.

B. Malware Detection

In the malware detection experiment, we compare the ER-Caps with the baseline CapsNet and with different CNN models: DensNet121, ResNet50, and VGG16.

TABLE XIII
MODELS PERFORMANCES ON MALWARE DETECTION.

Model	Precision %	Recall %	F1-score %	Parameters (M)
VGG16	96.10	96.76	96.43	14.84
DensNet121	99.24	99.38	99.31	7.30
ResNet50	99.24	99.0	99.12	24.11
Baseline CapsNet	98.86	98.75	98.81	5.37
ER-Caps	99.24	99.38	99.31	4.5

For malware detection, the last layer of DensNet121, ResNet50 and VGG16 consists of two neurons representing a binary classification (malicious or benign). The same trend exists for the baseline CapsNet and the ER-Caps; the Class-Caps layer contains two class capsules. It was observed from Table XIII that ER-Caps showed competitive results compared to the other models; it outperformed all other models across all classification metrics, achieving a precision of 99.24%, which indicates that ER-Caps accurately identifies positive predictions. It is conservative in classifying samples as malware, reducing the probability of false alerts. This is particularly crucial in applications related to security, where reducing false positives is frequently a high concern. DenseNet121 has a performance similar to ER-Caps; however, ER-Caps is more efficient, has fewer parameters, and needs 2.8 million fewer parameters compared with DensNet121.

V. CONCLUSION & PERSPECTIVES

Detecting and classifying malware is a crucial task to protect the computer's systems, digital assets, and data against malicious attacks. Several modern antivirus solutions depend on deep learning approaches, which have demonstrated good performance in analyzing malware. In this paper, we study the application of a deep learning architecture based on residual blocks and CapsNet to malware detection and classification using byteplot malware images. The experimental results confirm that ER-Caps has great performance when dealing with imbalanced malware classification tasks, achieving an average weighted F1-score of 91.97%, 94.02%, and 98.43% on the VIRUS-MNIST, the MMCC, and the Malimg datasets respectivelyly. Furthermore, the model shows comparable performance on the MMCC dataset compared to other works proposed in the literature without any enhanced training strategies. Moreover, the ER-Caps proves its ability to detect malware and can reach a 99.31% F1-score with a small number of training parameters compared with the other deep learning networks.

Future work will be focused on adapting the model to process the input image of any size to improve our models flexibility. In addition, model compression to enable deployment on edge devices would be a promising area for future research.

REFERENCES

[1] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *AI 2016: Advances in Artificial Intelligence: 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5-8, 2016, Proceedings 29*. Springer, 2016, pp. 137–149.

[2] M. M. Andreas Marx, Guido Habicht. (2023-12-07) Malware statistics & trends report — av-test. The AV-TEST Institute, The Independent IT-Security Institute. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[3] A. Bensaoud, N. Abudawaood, and J. Kalita, "Classifying malware images with convolutional neural network models," *International Journal of Network Security*, vol. 22, no. 6, pp. 1022–1031, 2020.

[4] J. Saxe and H. Sanders, *Malware data science: attack detection and attribution*. No Starch Press, 2018.

[5] C. Guarnieri, M. Schloesser, J. Bremer, and A. Tanasi, "Cuckoo sandbox-open source automated malware analysis," *Black Hat USA*, 2013.

[6] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, pp. 1–12, 2017.

[7] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *The 5th Conference on Information and Knowledge Technology*. IEEE, 2013, pp. 113–120.

[8] H. Babbar, S. Rani, D. K. Sah, S. A. AlQahtani, and A. Kashif Bashir, "Detection of android malware in the internet of things through the k-nearest neighbor algorithm," *Sensors*, vol. 23, no. 16, p. 7256, 2023.

[9] R. S. Arslan and A. H. Yurttakal, "K-nearest neighbour classifier usage for permission based malware detection in android," *Icontech International Journal*, vol. 4, no. 2, pp. 15–27, 2020.

[10] T. Singh, F. Di Troia, V. A. Corrado, T. H. Austin, and M. Stamp, "Support vector machines and malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 12, pp. 203–212, 2016.

[11] F. Shang, Y. Li, X. Deng, and D. He, "Android malware detection method based on naive bayes and permission correlation algorithm," *Cluster Computing*, vol. 21, no. 1, pp. 955–966, 2018.

[12] J. Pang and J. Bian, "Android malware detection based on naive bayes," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2019, pp. 483–486.

[13] L. Sayfullina, E. Eirola, D. Komashinsky, P. Palumbo, Y. Miche, A. Lendasse, and J. Karhunen, "Efficient detection of zero-day android malware using normalized bernoulli naive bayes," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 198–205.

[14] A. Zulkifli, I. R. A. Hamid, W. M. Shah, and Z. Abdullah, "Android malware detection based on network traffic using decision tree algorithm," in *Recent Advances on Soft Computing and Data Mining: Proceedings of the Third International Conference on Soft Computing and Data Mining (SCDM 2018), Johor, Malaysia, February 06-07, 2018*. Springer, 2018, pp. 485–494.

[15] H.-D. Pham, T. D. Le, and T. N. Vu, "Static pe malware detection using gradient boosting decision trees algorithm," in *Future Data and Security Engineering: 5th International Conference, FDSE 2018, Ho Chi Minh City, Vietnam, November 28–30, 2018, Proceedings 5*. Springer, 2018, pp. 228–236.

[16] A. Utku, İ. A. Doğru, and M. A. Akcayol, "Decision tree based android malware detection system," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2018, pp. 1–4.

[17] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent vision-based malware detection and classification using deep random forest paradigm," *IEEE Access*, vol. 8, pp. 206 303–206 324, 2020.

[18] S. B. Atitallah, M. Driss, and I. Almomani, "A novel detection and multi-classification approach for iot-malware using random forest voting of fine-tuning convolutional neural networks," *Sensors*, vol. 22, no. 11, p. 4302, 2022.

[19] S. Yoo, S. Kim, S. Kim, and B. B. Kang, "Ai-hydra: Advanced hybrid approach using random forest and deep learning for malware classification," *Information Sciences*, vol. 546, pp. 420–435, 2021.

[20] G. Conti, E. Dean, M. Sinda, and B. Sangster, "Visual reverse engineering of binary and data files," in *International Workshop on Visualization for Computer Security*. Springer, 2008, pp. 1–17.

[21] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

[22] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *Advances in neural information processing systems*, vol. 30, 2017.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] E. Alaoui-Elfels, T. Gadi *et al.*, "Er-caps: Elu residual capsule network for complex images classification." *International Journal of Intelligent Engineering & Systems*, vol. 16, no. 1, 2023.

[25] K. Kosmidis and C. Kalloniatis, "Machine learning and images for malware detection and classification," in *Proceedings of the 21st Pan-Hellenic Conference on Informatics*, 2017, pp. 1–6.

[26] A. S. Bozkir, A. O. Cankaya, and M. Aydos, "Utilization and comparision of convolutional neural networks in malware recognition," in *2019 27th signal processing and communications applications conference (SIU)*. IEEE, 2019, pp. 1–4.

[27] M. Jayasudha, A. Shaik, G. Pendharkar, S. Kumar, B. Muhesh Kumar, and S. Balaji, "Comparative analysis of imbalanced malware byteplot image classification using transfer learning," in *International Conference on Power Engineering and Intelligent Systems (PEIS)*. Springer, 2023, pp. 313–324.

[28] X. Wang, J. Liu, and X. Chen, "Microsoft malware classification challenge (big 2015)," https://kaggle.com/competitions/malware-classification, 2015.

[29] A. P. Tuan, A. Phuong, N. V. Thanh, and T. Van, "Malware detection pe-based analysis using deep learning algorithm dataset," 2018.

[30] M. Hassan, M. Eid, H. Elnems, E. Ahmed, E. Mesak, and P. Branco, "Detecting malicious .net files using CLR header features and machine learning," in *36th Canadian Conference on Artificial Intelligence, Canadian AI, CANAI 2023, Montreal, Canada, June 5-9, 2023, Proceedings*. Canadian Artificial Intelligence Association, 2023. [Online]. Available: https://doi.org/10.21428/594757db.88040587

[31] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska, "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, vol. 9, pp. 5371–5396, 2020.

[32] J. Cheng, R. Xu, X. Tang, V. S. Sheng, C. Cai *et al.*, "An abnormal network flow feature sequence prediction approach for ddos attacks detection in big data environment," *Computers, materials & continua*, vol. 55, no. 1, pp. 95–119, 2018.

[33] G. Liang, J. Pang, and C. Dai, "A behavior-based malware variant classification technique," *International Journal of Information and Education Technology*, vol. 6, no. 4, p. 291, 2016.

[34] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, "Behavior-based features model for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 12, pp. 59–67, 2016.

[35] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on api call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 659101, 2015.

[36] S. Anju, P. Harmya, N. Jagadeesh, and R. Darsana, "Malware detection using assembly code and control flow graph optimization," in *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, 2010, pp. 1–4.

[37] M. Z. Shafiq, S. A. Khayam, and M. Farooq, "Embedded malware detection using markov n-grams," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2008, pp. 88–107.

[38] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of iot malware and detection methods based on static features," *ICT express*, vol. 6, no. 4, pp. 280–286, 2020.

[39] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 2014, 2014.

[40] 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.

[41] K. M. PHILIP O'KANE, SAKIR SEZER, "Obfuscation: The hidden malware," *IEEE Security & Privacy*, 2011.

[42] L. Cavallaro, P. Saxena, and R. Sekar, "On the limits of information flow techniques for malware analysis and containment," in *International conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008, pp. 143–163.

[43] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.

[44] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *Journal of Computer Virology and Hacking Techniques*, vol. 15, pp. 15–28, 2019.

[45] Y. H. Choi, B. J. Han, B. C. Bae, H. G. Oh, and K. W. Sohn, "Toward extracting malware features for classification using static and dynamic analysis," in *2012 8th International Conference on Computing and Networking Technology (INC, ICCIS and ICMIC)*. IEEE, 2012, pp. 126–129.

[46] M. Eskandari, Z. Khorshidpour, and S. Hashemi, "To incorporate sequential dynamic features in malware detection engines," in *2012 European Intelligence and Security Informatics Conference, EISIC 2012, Odense, Denmark, August 22-24, 2012*, N. Memon and D. Zeng, Eds. IEEE Computer Society, 2012, pp. 46–52. [Online]. Available: https://doi.org/10.1109/EISIC.2012.57

[47] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[48] M. R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. Netw. Comput. Appl.*, vol. 36, pp. 646–656, 2013.

[49] X. Ma, Q. Biao, W. Yang, and J. Jiang, "Using multi-features to reduce false positive in malware classification," *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, pp. 361–365, 2016.

[50] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *CISIS/ICEUTE/SOCO Special Sessions*, 2012.

[51] D. Ucci, L. Aniello, and R. Baldoni, "Survey on the usage of machine learning techniques for malware analysis," *Computers & Security,*, vol. 81, pp. 123–147, 2017.

[52] K. Hughes and Y. Qu, "A theoretical model: Using logistic regression for malware signature based detection," in *The 10th International Conference on Dependable, Autonomic, and Secure Computing (DASC-2012)*, 2012.

[53] S. Kilgallon, L. D. L. Rosa, and J. Cavazos, "Improving the effectiveness and efficiency of dynamic malware analysis with machine learning," *2017 Resilience Week (RWS)*, pp. 30–36, 2017.

[54] K. Kancherla, J. Donahue, and S. Mukkamala, "Packer identification using byte plot and markov plot," *Journal of Computer Virology and Hacking Techniques*, vol. 12, pp. 101–111, 2016.

[55] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. IEEE, 2013, pp. 40–44.

[56] H. Naeem, B. Guo, and M. R. Naeem, "A light-weight malware static visual analysis for iot infrastructure," in *2018 International conference on artificial intelligence and big data (ICAIBD)*. IEEE, 2018, pp. 240–244.

[57] M. N. Yusoff and A. Jantan, "Optimizing decision tree in malware classification system by using genetic algorithm," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 1, p. 694–713, 2011.

[58] C. D. Morales-Molina, D. Santamaria-Guerrero, G. Sanchez-Perez, H. Perez-Meana, and A. Hernandez-Suarez, "Methodology for malware classification using a random forest classifier," in *2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*. IEEE, 2018, pp. 1–6.

[59] M. S. Alam and S. T. Vuong, "Random forest classification for detecting android malware," in *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*. IEEE, 2013, pp. 663–669.

[60] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[61] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of cnn architectures (imcec)," *Computers & Security*, vol. 92, p. 101748, 2020.

[62] J. Kiger, S.-S. Ho, and V. Heydari, "Malware binary image classification using convolutional neural networks," in *Int. Conf. Cyber Warf. Secur*, vol. 17, 2022, pp. 469–478.

[63] R. Chaganti, V. Ravi, and T. D. Pham, "Image-based malware representation approach with efficientnet convolutional neural networks for effective malware classification," *Journal of Information Security and Applications*, vol. 69, p. 103306, 2022.

[64] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. d. Geus, "Malicious software classification using vgg16 deep neural network's bottleneck features," in *Information Technology-New Generations: 15th International Conference on Information Technology*. Springer, 2018, pp. 51–59.

[65] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, "Malicious software classification using transfer learning of resnet-50 deep neural network," in *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 2017, pp. 1011–1014.

[66] W. Li, R. Zhang, and Q. Wen, "A malicious code variants detection method based on self-attention," in *Proceedings of the 2020 6th International Conference on Computer and Technology Applications*, 2020, pp. 51–56.

[67] M. Nisa, J. H. Shah, S. Kanwal, M. Raza, M. A. Khan, R. Damaševičius, and T. Blažauskas, "Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features," *Applied Sciences*, vol. 10, no. 14, p. 4966, 2020.

[68] R. Damaševičius, A. Venčkauskas, J. Toldinas, and Š. Grigaliūnas, "Ensemble-based classification using neural networks and machine learning models for windows pe malware detection," *Electronics*, vol. 10, no. 4, p. 485, 2021.

[69] O. El Alaoui-Elfels and T. Gadi, "From auto-encoders to capsule networks: A survey," in *E3S web of conferences*, vol. 229. EDP Sciences, 2021, p. 01003.

[70] R. Katarya and Y. Arora, "Study on text classification using capsule networks," in *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE, 2019, pp. 501–505.

[71] P. Afshar, K. N. Plataniotis, and A. Mohammadi, "Capsule networks for brain tumor classification based on mri images and coarse tumor boundaries," in *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2019, pp. 1368–1372.

[72] E. Butun, O. Yildirim, M. Talo, R.-S. Tan, and U. R. Acharya, "1d-cadcapsnet: One dimensional deep capsule networks for coronary artery disease detection using ecg signals," *Physica Medica*, vol. 70, pp. 39–48, 2020.

[73] X. Zhang, K. Wu, Z. Chen, and C. Zhang, "Malcaps: A capsule network based model for the malware classification," *Processes*, vol. 9, no. 6, p. 929, 2021.

[74] S.-w. Wang, G. Zhou, J.-c. Lu, and F.-j. Zhang, "A novel malware detection and classification method based on capsule network," in *Artificial Intelligence and Security: 5th International Conference, ICAIS 2019, New York, NY, USA, July 26-28, 2019, Proceedings, Part I 5*. Springer, 2019, pp. 573–584.

[75] A. Çayır, U. Ünal, and H. Dağ, "Random capsnet forest model for imbalanced malware type classification task," *Computers & Security*, vol. 102, p. 102133, 2021.

[76] Z. Wang, W. Han, Y. Lu, and J. Xue, "A malware classification method based on the capsule network," in *Machine Learning for Cyber Security: Third International Conference, ML4CS 2020, Guangzhou, China, October 8–10, 2020, Proceedings, Part I 3*. Springer, 2020, pp. 35–49.

[77] B. Zou, C. Cao, L. Wang, and F. Tao, "Dacn: malware classification based on dynamic analysis and capsule networks," in *International Conference on Frontiers in Cyber Security*. Springer, 2021, pp. 3–13.

[78] Malevis dataset. Multimedia Information Lab of Hacettepe University Computer Engineering, COMODO Inc. Accessed: January 20, 2024. [Online]. Available: https://web.cs.hacettepe.edu.tr/ selman/malevis/

[79] D. A. Noever and S. E. M. Noever, "Virus-mnist: A benchmark malware dataset," *CoRR*, vol. abs/2103.00602, 2021.

[80] O. El Alaoui-Elfels and T. Gadi, "Emg-capsnet: Elu multiplication gate capsule network for complex images classification," in *International Conference on Soft Computing and Pattern Recognition*. Springer, 2021, pp. 97–108.

[81] E. Alaoui-Elfels, T. Gadi *et al.*, "Tg-capsnet: Two gates capsule network for complex features extraction." *Journal of Information Assurance & Security*, vol. 16, no. 5, 2021.

[82] M. Sewak, S. K. Sahay, and H. Rathore, "Comparison of deep learning and the classical machine learning algorithm for the malware detection," in *2018 19th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD)*. IEEE, 2018, pp. 293–296.

[83] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.

[84] B. Zou, C. Cao, L. Wang, S. Fu, T. Qiao, and J. Sun, "FACILE: A capsule network with fewer capsules and richer hierarchical information for malware image classification," *Comput. Secur.*, vol. 137, p. 103606, 2024.

[85] M. Jayasudha, A. Shaik, G. Pendharkar, S. Kumar, B. Muhesh Kumar, and S. Balaji, "Comparative analysis of imbalanced malware byteplot image classification using transfer learning," in *International Conference on Power Engineering and Intelligent Systems (PEIS)*. Springer, 2023, pp. 313–324.