A few-shot malware classification approach for unknown family recognition using malware feature visualization

Conti, Mauro; Khandhar, Shubham ; Vinod, P.

DOI
10.1016/j.cose.2022.102887

Publication date
2022

Document Version
Final published version

Published in
Computers and Security

Citation (APA)
Conti, M., Khandhar, S., & Vinod, P. (2022). A few-shot malware classification approach for unknown family recognition using malware feature visualization. *Computers and Security*, *122*, 1-16. Article 102887. https://doi.org/10.1016/j.cose.2022.102887

Computers & Security 122 (2022) 102887

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

A few-shot malware classification approach for unknown family recognition using malware feature visualization

Mauro Conti [a,b], Shubham Khandhar [b], P. Vinod [c,*]

[a] Department of Mathematics, University of Padua, Padua, Italy
[b] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, the Netherlands
[c] Department of Computer Applications, Cochin University of Science & Technology, Cochin, Kerala, India

ARTICLE INFO

Article history:
Received 30 January 2022
Revised 21 July 2022
Accepted 18 August 2022
Available online 20 August 2022

Keywords:
Malware classification
Few-shot learning
Siamese neural networks
Deep neural networks
GEM Image
Malware visualization

ABSTRACT

With the ever-increasing threat of malware attacks, building an effective malware classifier to detect malware promptly is of utmost importance. Malware visualization approaches and deep learning techniques have proven effective in classifying sophisticated malware from benchmark datasets. A major problem with traditional deep learning classifier is the need to re-train the classifier when a new malware family emerges. In this paper, we propose few-shot classification techniques which allows us to classify malware based on a few instances and without the need for re-training the classifier for novel malware families. We also propose a novel malware visualization technique that can represent a malware binary as a 3-channel image. We experiment with two distinct few-shot learning architectures namely CSNN (Convolutional Siamese Neural Network) and Shallow-FS (Shallow Few-Shot). CSNN is more suitable when scarce data is available for training, otherwise Shallow-FS can be used to achieve better performance. Our architectures outperforms state of the art few-shot learning approaches and achieves high accuracy in traditional malware classification. Our experiments show our models' ability to classify recent and novel malware families from just a few instances with high accuracy.

1. Introduction

The ever-increasing risk of malware (**mal**icious soft**ware**) attacks have been a significant threat to internet users around the globe. Currently, malware is one of the primary attack vectors used by cybercriminals to perform malicious activities. The Mcafee ATR Threat Reports (2021) showed an immense increase in Powershell threats, MacOS malware, Office malware, Mobile malware, Ransomware, and Linux malware in the second half of 2020. According to the AV-test statistics report (AV Test malware statistics, 2021), they discovered approximately 100 million new malware files in the first half of 2021. As reported by the Kaspersky Security Network (IT threat evolution, 2021), they observed new attempts to run money stealing malware on the computers of 119,252 unique users in Q2 of 2021. These statistics support the fact that malware is a growing threat to Internet users. It is important to note that with the steady increase in sheer number of malware, their families and their variants are also constantly evolving according to the aforementioned reports.

The functionalities and capabilities of a malware can vary depending upon various factors such as the intended platform for malware, its types, its family, and other malware characteristics. Adware, Trojan, Backdoor, Ransomware, Spyware, Worm, etc., are a few examples of types of malware. Malware types can be further divided into families and their variants based on various factors such as malware codebase, malware development groups, and many more (SANS Webcast Recap, 2020). To avoid detection, malware authors use a variety of obfuscation techniques such as dead-code insertion, instruction reordering, and control flow flattening (Alrabaee et al., 2018). The use of obfuscation techniques in combination with frequent updates to the malware codebase makes it very challenging to accurately classify malware into its families.

Malware poses massive security risks to governments, businesses and individual users. There exist multiple strategies to mitigate malware attacks and accurate malware classification is an integral part of these strategies. Security analysts and researchers analyse malware files to understand their behavioural characteristics and purpose, which helps them build better defenses against other malware files from the same family. There are mainly two distinct types of malware analysis: *static analysis* and *dynamic analysis*. Static analysis involves analyzing the malware binary content without executing it. On the other hand, dynamic analysis involves

* Corresponding authors.
E-mail addresses: conti@math.unipd.it (M. Conti), S.C.Khandhar@student.tudelft.nl (S. Khandhar), vinod.p@cusat.ac.in (P. Vinod).

https://doi.org/10.1016/j.cose.2022.102887
0167-4048/

analyzing the malware behaviour while/after executing it in memory. Static analysis is faster than dynamic analysis, but it is not resilient against sophisticated code obfuscation techniques. On the other hand, dynamic analysis is often unaffected by code obfuscation and polymorphic malware (Gibert et al., 2020b) but is slower in comparison.

Traditionally, malware detection/classification is performed using signature-based or heuristic-based methods. Signature-based methods deploy a signature for distinct malware families and variants, which acts as a prototype and allows newly discovered malware files to be accordingly classified. Heuristic-based methods, on the other hand, uses rules and byte patterns created by industry experts and analysts from existing malware data to classify new malware (Ye et al., 2017). These approaches mostly fall under the category of static malware analysis. In the last few years, a static malware analysis technique known as Malware visualization introduced by Nataraj et al. (2011) has prevailed in solving the problem of malware classification. Many recent works, as mentioned in Section 2.1, have used this technique to tackle the problem of malware classification.

Malware visualization is a technique that consists of representing the contents of a malware binary in some form as an image. Traditionally, the raw bytes of the malware binary are read as 8-bit unsigned integers and stored into a vector. This vector is reshaped into a matrix and can then be visualized as a grayscale image. After carefully performing multiple experiments, we noted that certain features extracted from the malware were more accurate in classifying malware than raw bytes. Instead of visualizing the raw contents of the malware binary, as proposed in previous research works (Dai et al., 2018; Nataraj et al., 2011; Vasan et al., 2020a; 2020b), we visualize the various features extracted from the malware binary which are inspired by recent malware classification approaches (Han et al., 2015; Xiao et al., 2020; Yuan et al., 2020).

Furthermore, we investigate the use of a shallow Convolutional Neural Network (CNN) (LeCun et al., 1995) architecture in combination with few-shot learning to recognize unknown malware classification. The Few-shot learning approach was first proposed by Fei-Fei et al. (2006). Few-shot models learn classification from a few labelled instances from each given class (Wang et al., 2020). The idea behind our approach is to provide a testbed for a deep learning model to classify malware feature images when provided with a few instances of each class and that some of these malware families were unknown to the classifier while training. Formally, we use the term *unknown family/class* throughout this paper for families/classes that are unknown to the classifier during its training and are only introduced during evaluation. In this paper, we use the few-shot classification terms such as query image/instance, support set, similarity metric, etc. as defined in other well-known few-shot learning papers (Chen et al., 2019; Tran et al., 2019; Wang et al., 2020). We implement the few-shot learning technique using Convolutional Siamese Neural Network (CSNN) (Bromley et al., 1993; Koch et al., 2015) and the Shallow-FS architecture on three distinct PE malware datasets.

The summary of the contributions of this paper are mentioned below:

- We performed comprehensive experiments on benchmark PE datasets such as Malimg and Microsoft BIG 2015. Additionally, we compiled our own dataset, namely the MalwareBazaar (Malbaz) dataset, containing recent malware collected in mid-2021 to thoroughly validate the effectiveness of our classifier.
- We propose a novel malware visualization technique, namely the GEM image. The GEM coloured 3-channel image is formed by fusing the Markov image, entropy graph image and gray-level matrix image. Our experiments show that the GEM images outperform the standard malware visualization techniques such as grayscale images and colour-mapping grayscale images. We also introduce a novel way to visualize GLCM features by rearranging and visualizing the Gray-Level Co-occurrence Matrix, instead of extracting 2nd order statistical features from it.
- Our CSNN architecture combined with the GEM image outperforms the state of art few-shot classification approaches for the benchmark PE malware datasets. CSNN achieves an unknown family classification accuracy of 96.21%, 94.99% and 93.42% for the three aforementioned datasets, when trained on approximately 10% of the dataset and evaluated on the rest.
- To the best of our knowledge, we are the first to experiment with malware classification under the standard evaluation setting for few-shot classification as described by Chen et al. (2019) in their paper. Inspired by their work, we use the Shallow-FS model to perform 10-shot unknown family classification which yielded an accuracy of 98.26%, 88.68% and 97.65% for the three datasets. This method also outperforms the few-shot classification approaches for the benchmark PE malware datasets.

Section 2 provides a comprehensive discussion of the related work in the domain of malware classification and visualization. Section 3 provides a detailed description of our proposed methodologies including the datasets, feature extraction techniques, GEM image construction and model architectures used in this paper. Section 4 continues with an individual discussion for each experiment we performed and its results in detail. Section 5 discusses the results of the experiments results and we try to explain our results and draw comparisons with other state of the art classifiers. Section 6 concludes our paper and provides suggestions and a few ideas that can enhance and continue our malware classification approach.

2. Related work

2.1. Malware detection/classification based on malware visualized images and deep learning

The concept of malware visualization was first proposed by Nataraj et al. (2011) that targets at representing the bytes of a malware as pixels of an image. The malware visualized as an image can be fed to various CNN architectures for classification. They were the first to classify malware using visualization techniques. They represented binary files as grayscale images and utilized similarity computations. They also used GIST features to extract gradient properties and these features were provided to a K-Nearest Neighbor (KNN) classifier. Makandar and Patrot (2017) used the visualisation technique in combination with Support Vector Machines (SVM) classifier. They used feature processing techniques such as Gabor Wavelet, GIST and Discrete Wavelet Transform to construct an effective texture feature vector of an image. Ni et al. (2018) and Qiao et al. (2019) used specific algorithms, such as SimHash and Word2Vec (Mikolov et al., 2013) respectively, on malware disassembly to represent it as an image and further classify using deep CNNs.

Xiao et al. (2020) used the structural entropy to visualize malware binaries as entropy graphs. These entropy graphs are fed to a deep CNN to extract features and an SVM classifier is used for classification. Vu et al. (2020) proposed an approach that targets pixel encoding and arranging bytes from malware binaries into images for malware detection. These images have statistical and syntactic artifacts, for example, entropy or strings, and their pixels are filled with space-filling curves. Vasan et al. (2020a,b) proposed two distinct approaches for malware visualized image classification. Firstly, they proposed an ensemble CNN-based architecture for obfuscated and un-obfuscated malware detection. Sec-

ondly, they presented a method that converts raw malware binaries into colour images and a fine-tuned CNN architecture for classification. In the second approach, they used transfer learning for better performance and data augmentation for tackling class imbalance. All of the approaches mentioned above achieved high accuracies for classifying malware.

2.2. Malware detection/classification using few-shot learning

The concept of Few-shot learning was first proposed by Fei-Fei et al. (2006). They explored a Bayesian implementation of the idea of taking advantage of previously learned knowledge instead of learning from scratch. Hsiao et al. (2019) used Siamese neural networks to classify malware images. They preprocess their datasets using malware visualization and average hash algorithm. They concluded that their approach outperformed baseline methods such as nearest neighbor and random guessing. Bai et al. (2020) also proposed the use of Siamese neural networks to classify Android malware. They used the Siamese neural network as a feature extractor to generate feature embeddings and attached it to a Multi-layer Perceptron (MLP) to classify the generated embeddings. Rong et al. (2021) converted the network traffic data generated by malware variants into grayscale images. These images are used as input to a prototype-based few-shot learning model. They achieved a very high accuracy and concluded that their method is universal and robust in detecting malware variants in network environments.

Wang et al. (2021) proposed a meta-learning based few-shot learning technique to classify novel malware families. They use API invocation sequences from dynamic analysis of malware. Their method achieved a high accuracy in a 5-way classification task on a Virus Share and APIMDS dataset. Tran et al. (2019) used well-known Meta-learning approaches such as Matching Network and Prototypical Network to classify malware visualized images. They performed experiments on benchmark PE malware datasets such as Malimg and Microsoft BIG 2015. They performed 1-shot and 4-shot malware classification experiments and achieved higher accuracies than previous works on the PE benchmark datasets. To the best of our knowledge, the last discussed approach is the only work that showed the effectiveness of static analysis features and few-shot classification, especially meta-learning, on PE malware datasets.

2.3. Malware detection/classification based on alternative approaches

Here we discuss the state-of-the-art research in the domain of malware detection/classification based on approaches that do not use malware visualization, deep learning or few-shot learning. Nataraj et al. (2010) employed an SVM to classify packed and unpacked executables based on bigram features. Burguera et al. (2011) built an Android malware detector that leverages dynamic analysis to discriminate between malicious and benign apps. Their method was effective in isolating malware and alerting application consumers that have already downloaded malware. Natani and Vidyarthi (2013) proposed a method for detecting malware that uses the API function frequency as a feature vector for categorizing malicious files. Chuang and Wang (2015) also proposed a model for malware detection based on SVM classifier and API calls.

Gibert et al. (2020a) introduced a framework for malware classification that combines distinct features to discover relationships between different modalities. Their multi-modal approach maximizes the benefits of multiple feature types while accurately reflecting the characteristics of malware executables and classification performance. Y. Ki et al. Ki et al. (2015) proposed a technique that used DNA sequence alignment algorithms to extract common API call sequence patterns of malicious functions from various types of malware. Their experiments concluded that their approach resulted in almost perfect classification on their own PE malware dataset. Pascanu et al. (2015) proposed an approach that learns the language of malware communicated through executed instructions and extracts robust, temporal domain properties, akin to natural language modelling. Their model was effective and outperformed classic Recurrent Neural Networks (RNN) in majority of their experiments.

Our work differs from the previously mentioned works, as we propose the classification of malware from the benchmark PE datasets using different few-shot learning approaches. To the best of our knowledge, we use shallow architectures such as CSNN and shallow-FS for few-shot classification which have not been used to classify PE malware before.

Table 1
Malimg Dataset.

Family code	Family	Number of samples
A1	Instantaccess	431
A2	Dialplatform.B	177
A3	Autorun.K	106
A4	Dontovo.A	162
A5	Lolyda.AA1	213
A6	Lolyda.AT	159
A7	Adialer.C	122
A8	Fakerean	381
A9	Rbot!gen	158
A10	Allaple.A	2949
A11	VB.AT	408
A12	Yuner.A	800
A13	Malex.gen!J	136
A14	Agent.FYI	116
A15	Skintrim.N	80
A16	Obfuscator.AD	142
A17	Lolyda.AA2	184
A18	Lolyda.AA3	123
A19	Wintrim.BX	97
A20	Swizzor_C2LOP	606
A21	Allaple.L	1591
A22	Alueron.gen!J	198

Table 2
Microsoft BIG 2015 Dataset.

Family code	Family	Number of samples
B1	Ramnit	1541
B2	Lollipop	2478
B3	Kelihos_v3	2942
B4	Vundo	475
B5	Simda	42
B6	Tracur	751
B7	Kelihos_v1	398
B8	Obfuscator.ACY	1228
B9	Gatak	1013

3. Proposed method

In this section, we discuss the methodologies and individual components of our proposed few-shot classification approach. Fig. 1 depicts our proposed approach and its components.

3.1. Datasets

To validate our approach and create an effective malware classifier, it is necessary to use well-known state of art malware datasets. We use three distinct datasets, namely the Malimg dataset (refer Table 1), Microsoft BIG 2015 (refer Table 2) dataset and MalwareBazaar dataset (refer Table 3).

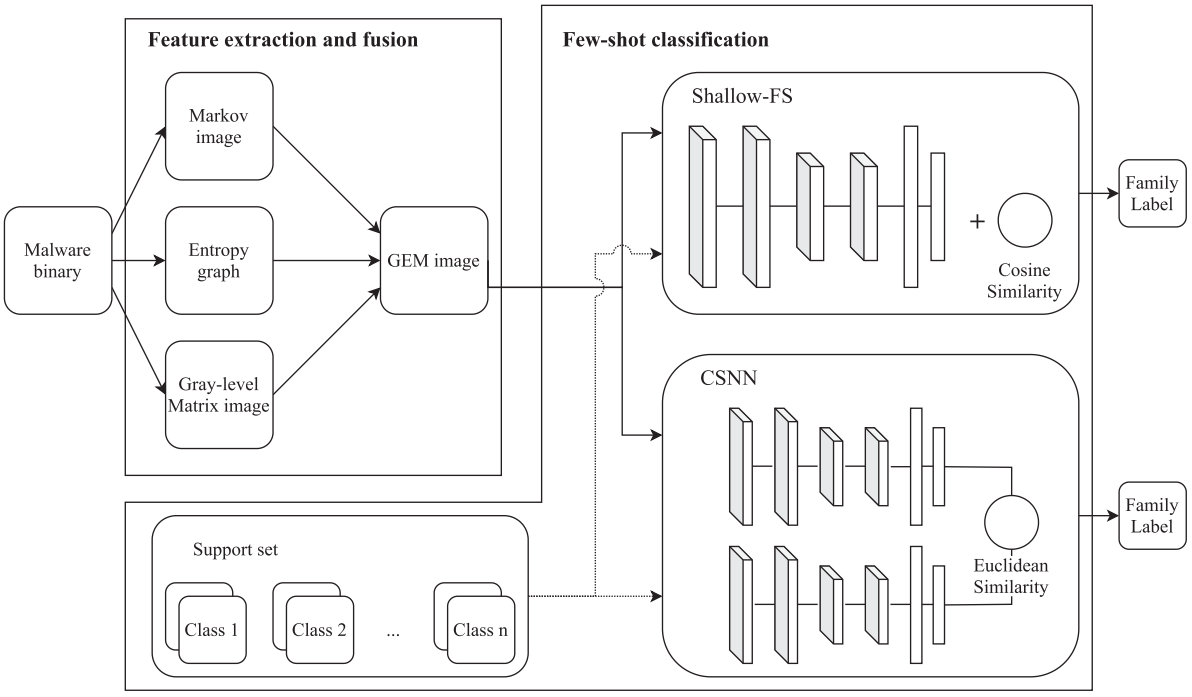


Fig. 1. Our proposed approach shows the process of extracting three distinct features from a malware binary and fusing them into a 3-channel GEM image. This GEM image is then used as input to our few-shot classification models to predict the family label of the malware binary. The *Support set* mentioned in the figure is a collection of a fixed number of images randomly chosen from each class of the training set. These images are conceptually similar to a point of reference for the ML model.

Table 3
Malbaz Dataset.

Family code	Family	Number of samples
C1	CobaltStrike	592
C2	Trickbot	513
C3	njrat	606
C4	Cutwail	445
C5	AveMariaRAT	604
C6	GandCrab	146
C7	Quakbot	462
C8	IcedId	430
C9	CryptBot	133
C10	RedLineStealer	345
C11	Dridex	1659
C12	Emotet	1229
C13	Sodinokibi	233
C14	AgentTesla	679

3.1.1. Malimg dataset (Dataset-A)

Nataraj et al. (2011) introduced this dataset intending to explore signal and image processing techniques in the field of malware classification, and researchers have used it as a benchmark in various state-of-art research. In total, it contains 9339 malware samples, distributed among 25 malware families, represented in the form of grayscale images. Instead of using the 25-family Malimg dataset, we use the 22-family Malimg dataset as suggested by the authors (Nataraj et al., 2011). The authors suggest combining the variants of *Swizzor* and *C2LOP* families into one family. We also discovered certain threat reports (Microsoft Threat report, 2009; 2010) suggesting that *Swizzor* is an alias for *C2LOP* family.

3.1.2. Microsoft BIG 2015 dataset (Dataset-B)

This dataset was introduced by Microsoft and was part of a Kaggle competition hosted by Microsoft Ronen et al. (2021) in 2015. It is a massive dataset consisting of 21,741 malware samples. This dataset is further divided into two parts, a training set of 10,868 samples and a testing set of 10,873 samples. The training set is labelled, and the testing set was not publicly released. For this work, we only use the labelled training set (specifically, only the *.bytes* file), which consists of 9 malware families.

It is important to note that a lot of files in the dataset have a lot of noise as many bytes in those files are represented by "??" symbol. We remove the 8 files that only contain the "??" symbols.

3.1.3. Malbaz dataset (Dataset-C)

Malware is frequently evolving which requires a malware classifier to possess the ability to classify recently discovered malware. We collected malware executables in May of 2021 from a public malware repository - MalwareBazaar. We leveraged the MalwareBazaar API to compile a dataset containing 8076 samples distributed among 14 malware families. We leveraged the threat intelligence provided by third-party vendors on MalwareBazaar to eliminate any false positives. Throughout this paper, we refer to this dataset as the Malbaz dataset. Almost all previous works test the efficacy of their classifier on older benchmark datasets. In this paper, we test our classifiers on the benchmark datasets as well as the Malbaz dataset to better gauge the effectiveness and efficacy of our classifier.

3.2. Feature extraction

It is essential to extract relevant features from the malware images to build an effective malware classifier using malware feature visualization. Instead of using a very deep CNN to extract textural and structural features from a grayscale image of a malware binary, we experimented with various malware-domain specific features that have shown to be effective in previous research works (Han et al., 2015; Xiao et al., 2020; Yuan et al., 2020) and are mentioned below.

3.2.1. Markov images

This feature extraction method was introduced and discussed by Yuan et al. (2020). For this technique, we view the bytes of the malware binary as byte streams.

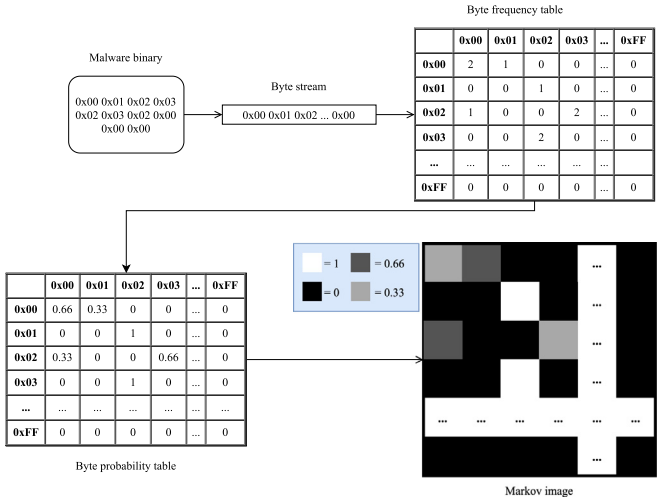


Fig. 2. An example of Markov image generation from a hypothetical malware binary. The malware binary is viewed as a byte stream and a Byte frequency table is generated based on the occurrence of each byte in the byte-steam. This Byte frequency table is then converted to a Byte probability table which when normalized results in the Markov image.

In order to generate a Markov image from a malware binary, a matrix of size 256×256 is generated and initialized with zeros. Formally, a sliding window of size 2 slides over the byte stream generating bi-grams out of the entire byte stream. For each bi-gram, we treat the first byte of the bi-gram as the row number, and we treat the second byte of the bi-gram as the column number and add a 1 at the respective position in the matrix. For example, a bi-gram 0x01 0x02 will result in a 1 being added to the cell corresponding row 1 and column 2.

$$BPT_{ij} = \begin{cases} \frac{BFT_{ij}}{S_i}, & \text{if } S_i \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$MI_{ij} = \frac{BPT_{ij} - \min(BFT[i])}{\max(BFT[i]) - \min(BFT[i])} \tag{2}$$

where

- BPT_{ij} represents the ith row and jth column of the Byte Probability Table. Similarly, BFT_{ij} and MI_{ij} represents the same values for the Byte Frequency Table and the Markov image respectively.
- S_i represents the sum of all values in the ith row.

After repeating this process for all bi-grams, we have a Byte frequency table with each cell representing the frequency of occurrence of each bi-gram. Further dividing each entry in the table by the sum of all entries in the respective row will transform it into Byte probability table which can be directly visualized as an image, namely the Markov image. As a result, an entry represented by row number x and column number y will be the conditional probability of a subsequent byte in the byte stream being y given the current byte is x. The generation of the Byte Probability Table and the Markov Image, as shown in Fig. 2, is formally described in Eq. (1) and (2). Fig. 3 depicts a few examples of Markov images from malware samples.

3.2.2. Entropy graph images

This feature extraction method was introduced by Han et al. (2015) and was further used and discussed by Xiao et al. (2020). In our approach, we slightly modify the approach by dividing the byte sequence of a malware binary into blocks of 128 bytes. The malware binary is padded with null bytes

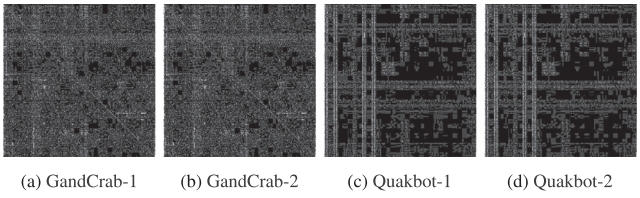


Fig. 3. Structural similarities and differences between Markov images. We enhanced the Markov images for better distinguishability and visual clarity.

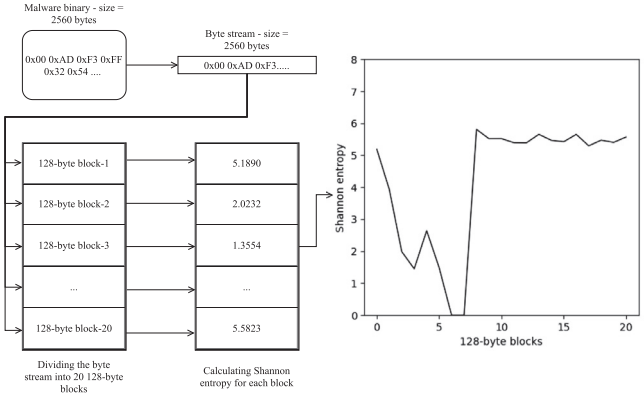


Fig. 4. An example of Entropy graph generation from a hypothetical malware binary of size 2560 bytes. The malware binary is divided into 128-byte blocks. After calculating the Shannon entropy of each block, these blocks are plotted in a graph forming the Entropy graph image.

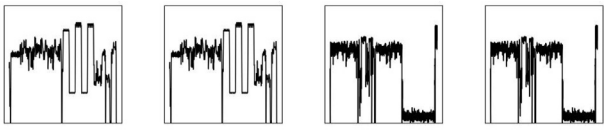


Fig. 5. Structural similarities and differences between Entropy Graph images.

to make the binary size a multiple of 128. Given that an executable contains n blocks of 128 bytes, we calculate the Shannon entropy of each block as mentioned in Eq. (3)

$$H(B_i) = -\sum_{k=0}^{255} P(b_{ki}) * log(P(b_{ki})) \tag{3}$$

where

- B_i stands for ith block and b_{ki} stands for kth byte in the ith block
- $H(B_i)$ is the Shannon entropy of ith 128-byte block of the byte sequence and $P(b_{ki})$ is the probability of kth byte in the ith block.

After we obtain the Shannon entropy for all 128-byte blocks, we plot these block entropies on a graph where the X-axis represents the 128-byte blocks, and the Y-axis represents the Shannon entropy of the respective block. All the plots generated are resized to a size of 256×256.

The resizing of the image allows us to use any malware binary regardless of its file size. Fig. 4 summarizes the Entropy graph generation process and Fig. 5 depicts a few examples of Entropy graph images.

3.2.3. Gray level matrix image

This feature extraction approach is one of the contributions of our work. Most previous works used the 2nd order statistical features extracted from the Gray-Level Co-occurrence Matrix (GLCM).

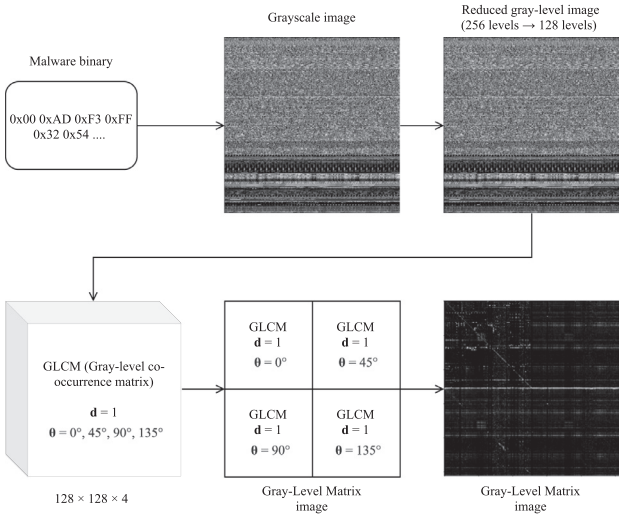
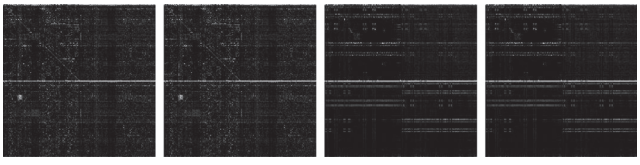


Fig. 6. An example of Stacked gray-level matrix generation from a hypothetical malware binary. The malware binary is visualized and then a GLCM is created from the grayscale image with reduced levels. This GLCM is then rearranged to form a Gray-level matrix image.



(a) GandCrab-1 (b) GandCrab-2 (c) Quakbot-1 (d) Quakbot-2

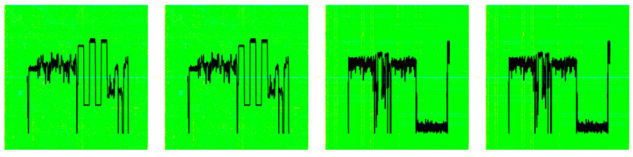
Fig. 7. Structural similarities and differences between Gray-level matrix images.

These features are one-dimensional, and thus, are not compatible with the input requirements of a CNN. Our solution to this problem was to use the GLCM and the textural and spatial information stored in it directly as an image, namely Gray Level Matrix Image, which is compatible with a 2-dimensional CNN.

To begin generating the Gray-level matrix images, we convert each malware binary to a grayscale image and then discretize the grayscale image reducing its levels from 256 to 128. The discretization speeds up the GLCM generation process significantly. A GLCM is calculated, from this 128-level grayscale image, for distance value of 1 and angle values of 0°, 45°, 90° and 135°. The resulting GLCM would be a 4-dimensional matrix of shape $128 \times 128 \times 1 \times 4$. Formally, for each distance and angle value, this matrix contains a sub-matrix of size 128×128 containing the textural and spatial information. As we only use one distance value, the calculated GLCM can be reshaped to $128 \times 128 \times 4$, and the sub-matrices of size 128×128 for all 4 angle values are separated. These 4 matrices are stacked as illustrated in Fig. 6, to form a final image of size 256×256. The pixel values of this image are normalized between 0 and 1 before inputting into a CNN. Fig. 6 summarizes the Gray-level Matrix image generation process and Fig. 7 depicts a few examples of Gray-level Matrix images.

3.3. GEM image construction

To achieve our aim of building an effective malware classification system and enhancing our system's performance, we experimented with all the feature extraction techniques mentioned previously. We generated feature images from malware binaries and attempted traditional malware classification using the Shallow-CNN model discussed in the subsequent section. After experimenting with different feature extraction techniques, we observed



(a) GandCrab-1 (b) GandCrab-2 (c) Quakbot-1 (d) Quakbot-2

Fig. 8. Structural similarities and differences between GEM images.

Table 4
Baseline model configuration. The abbreviation ReLU stands for Rectified Linear Unit.

Layer	Activation	Filters	Units	Kernel size
Conv2D	ReLU	128	-	(5, 5)
MaxPool2D	-	-	-	(2, 2)
Conv2D	ReLU	256	-	(3, 3)
MaxPool2D	-	-	-	(2, 2)
Flatten	-	-	-	-
Dense	ReLU	-	256	-
Dense	ReLU	-	128	-
Dense	Softmax	-	variable	-

mixed results, i.e. some feature extraction techniques performed better on one dataset, whereas others performed better on another dataset. The experimentation details are mentioned in Section 4.1.

To solve this problem and to achieve competitive performance with the state-of-the-art results across all the datasets, we decided to fuse the individual feature images into a single 3-channel image. The fusion process begins with an empty array of shape $3 \times 256 \times 256$. The Markov image, Entropy graph image and Gray-level matrix image are placed at each index of this array. Therefore, we refer to this 3-channel image as the GEM (Gray-level matrix image + Entropy graph image + Markov image) image in this work.

A significant advantage of having a 3-channel image is that it can be visualized as a RGB coloured image which can be used directly with state of art Transfer learning models. Fig. 8 depicts a few example of GEM images.

3.4. Traditional classification models

In this section, we describe the traditional CNN architectures used in this paper.

3.4.1. Shallow-CNN architecture

Our aim was to build an effective yet practically deployable malware classifier. We use a shallow CNN architecture that can be trained efficiently, can classify malware quickly and with a high accuracy. Thus, we experimented with a shallow CNN architecture, referred as the Shallow-CNN model (Khandhar, 2021). This model has two convolution layers, each of which is followed by a pooling layer, followed by two fully-connected layers. Table 4 shows the full configuration of the Shallow-CNN. In contrast to most state of art Transfer learning and Deep learning models, the training of the Shallow-CNN is quick.

3.5. Few-shot classification models

Scarcity of malware samples for malware families is a major problem when it comes to the domain of malware classification (Wang et al., 2021). Training a traditional malware classifier requires a large amount of data. Another limitation of traditional Deep-learning models is that they can only classify an instance into one of the classes the model was trained on.

In order for the model to classify data from a previously unknown class, it would require re-training using a significant

amount of data from this new class. To tackle these problems, we use two models (1) Convolutional Siamese Neural Network (CSNN) (2) Shallow Few-Shot (Shallow-FS). For each model, we discuss below the scenarios in which they can be more effective.

3.5.1. CSNN

The Siamese Neural Network was first conceptualized and proposed by Bromley et al. (1993). It has previously been used to classify malware by various researchers. After repeated experiments, we observed that CSNN can classify malware with high accuracy after only training it on a small percentage of the dataset and when there is an equal number of instances from each class in the training set. These observations are also supported by other previous research works (Bai et al., 2020; Hsiao et al., 2019). The reason behind this could be that the CSNN learns similarities and dissimilarities between pairs of training data instead of learning to map a dataset instance to a softmax activation.

The general structure of the CSNN can be visualized as two CNNs operating in parallel with shared weights and combined with a similarity metric (refer Fig. 10). The CNN used in our CSNN is the same as Shallow-CNN but with the last Softmax-activated layer removed. During backpropagation, the weights and parameters are updated simultaneously for both CNNs. The output of the last fully-connected layer of both the CNNs is provided as input to a Euclidean similarity metric which outputs a score between 0 and 1, which is formally described in Eqs. (4) and (5).

The Euclidean similarity used in the CSNN is described by the following equation:

$$d(P,Q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{4}$$

$$Euclidean\ similarity = \frac{1}{1 + d(P,Q)} \tag{5}$$

calculates the similarity score based on euclidean distance as shown in Segaran (2007) where

- d represents Euclidean distance and P, Q are the two feature vectors.
- p_i and q_i represent the ith element of the feature vectors P and Q respectively.

While training the CSNN, multiple pairs of images from the same family and different families are provided. After training the CSNN, in an ideal case, we expect the output of the CSNN to be 1 for similar images (malware images from the same family) and 0 for dissimilar images (malware images from different families). The branches of the CSNN can be visualized as an embedding function that embeds the input image in the embedding space. During its training, the CSNN learns to generate similar embeddings for similar input images and vice versa. In the end, the embeddings from the same class output a high Euclidean similarity in the embedding space and vice versa. We use the CSNN to perform one-shot malware classification, where the support set contains one image from each family we wish to classify. For every query instance, we provide the CSNN with a pair consisting of the query instance and each image from the support set. The family label of the support set image, which is contained in the pair that outputs the highest similarity score, is assigned to the query instance.

CSNN can tackle the data scarcity problem because it does not require large amounts of training data. It can also tackle the unknown class classification problem, under the condition that at least one instance of that class is available in the support set. As the CSNN learns the similarity between images, it is able to recognize the similarity between the query instance and the support set instance from the same unknown class.

3.5.2. Shallow-FS

This model was inspired by the *Baseline* classifier proposed by Chen et al. (2019). The authors in their work concluded that their simple *Baseline* model outperformed many meta-learning based few-shot classification approaches. The *Baseline* model consists of a pre-trained CNN-based feature extractor and a Multi-Layer Perceptron (MLP) with a single hidden layer and a softmax layer to output class probabilities. We experimented with their architecture and slightly simplified it for efficient performance. In our Shallow-FS, we completely remove the MLP and instead use a Cosine similarity metric directly on the feature vectors extracted by the CNN-based feature extractor. The CNN architecture we use here is the same as the Shallow-CNN architecture with the Softmax-activated layer removed. The advantage of this simplified Shallow-FS model over *Baseline* model is that no fine-tuning is required when unknown classes are added/removed from the support set of the classifier.

The structure of the Shallow-FS begins with the feature extractor from our Shallow-CNN model which maps an image to an embedding in the embedding space.

For the purpose of this work, we perform 10-shot classification with the Shallow-FS which implies selecting ten random instances from each class for the support set. Theoretically, it is possible to perform N-shot classification where N can be any number. It is also possible to perform variable-shot learning where a different number of images are chosen from each class to be in the support set, but it is outside the scope of this study. The embeddings of each class in the support set are then averaged to generate mean embeddings for each class in the support set. The Cosine similarity metric outputs a similarity score, between 0 and 1, for each pair consisting of a support set embedding and the query embedding. The family label of the support set image, which is contained in the pair that outputs the highest similarity score, is assigned to the query instance. Algorithm 1 describes the process of 10-shot classification in detail.

The training of the Shallow-FS feature extractor is conducted in a traditional manner. Formally, the Shallow-CNN model is trained with malware data and then the Softmax-activation layer is removed which results in a trained feature extractor for the Shallow-FS. The Shallow-FS model can tackle the data scarcity problem, under the practical observation that there exist malware families with sufficient data for training. Thus, the Shallow-FS feature extractor can be trained on this data and can be used to classify malware families with scarce data. In order to tackle the unknown class classification problem, it works in a similar manner to the CSNN. An appropriately trained feature extractor should generate similar embeddings for instances of the same family and vice versa.

The cosine similarity score is formally described by the following equation:

$$cosine\_similarity(X,Y) = \frac{X \cdot Y}{|X||Y|} \tag{6}$$

where X, Y are the two feature vectors and $\cdot$ represents the vector dot product.

3.6. Evaluation metrics

We mention all the standard performance metrics that we use to measure and evaluate the performance of our classifiers.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

$$Precision = \frac{TP}{TP + FP} \quad \text{and} \quad Recall = \frac{TP}{TP + FN} \tag{8}$$

Algorithm 1: Pseudocode for 10-shot classification with Shallow-FS.

Input : pre-trained *feature_extractor*, *test_sample*, 10-shot *support_set* of shape $n \times 10 \times 256 \times 256 \times 3$ where n is the number of classes

Output: The malware family label of the input GEM image *family_label*

```
/* We describe the pseudocode for generating the mean
   feature vectors for the support set. It is important
   to note that this needs to be performed only once
   until any classes are added/removed from the support
   set for classification                                */
num_classes ← n;
/* The array that will contain the similarity scores of
   all the query image and support set images' pairs.
   An index i represents the similarity score of the
   ith class                                             */
result_arr ← empty array of size num_classes;
/* The array that will contain the averaged feature
   vectors of all the support set images per class       */
mean_support_set ← empty array of size
num_classes × 256 × 256 × 3;
for i ← 0 to (num_classes − 1) do
    /* The array that contains the feature vectors
       produced by the Shallow-FS model for all 10
       images of the ith class                           */
    feature_vectors ← empty array of size 10 × 128;
    feature_vectors ← model.predict(support_set[i]);
    /* Average the feature vectors and store the mean
       feature vector in the ith index of the array      */
    mean_support_set [i] ← mean(feature_vectors , axis = 0);
end
/* Get the feature vector of the query image from the
   Shallow-FS                                            */
feature_vector_input ← model.predict(input_img);
for i ← 0 to (num_classes − 1) do
    /* Get the cosine similarity score between the query
       image feature vector and ith class support set
       mean feature vector, and store it at the ith
       index of the results_arr                          */
    result_arr [i] ← cosine_similarity(feature_vector_input ,
    mean_support_set [i]);
end
/* The family label is the index of the element in the
   array with the highest similarity score               */
family_label ← argmax(result_arr)
```

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{9}$$

There are two other metrics that we use in this paper:

Macro F1 score is defined as the unweighted mean of class-wise F1 scores.

Weighted F1 score is defined as the weighted mean of class-wise F1 scores. In other words, it is a modification of Macro F1 score that takes class imbalance into account. The mean is weighted by the number of instances in each class.

4. Classification experiments and results

This section describes all of our significant experiments. It is important to note that, due to the COVID-19 global pandemic, the availability and accessibility to powerful hardware was very limited. Thus, we were forced to perform experiments on personal laptops and cloud services such as Google Colaboratory. All of our models are implemented in *Keras* using a *Tensorflow* backend. The execution of training and the evaluation of the models was performed on Google Colaboratory running a Python 3.7 environment. We used the free Google Colaboratory instances to execute our code which adjusts the hardware allocation on the fly, providing no guarantees for resource availability (Google Colaboratory, 2021). The data preprocessing, feature extraction and GEM image generation was performed on an Apple Macbook Pro-2017. Ideally available resources on a free Google Colaboratory instance and an Apple Macbook Pro-2017 are mentioned in Table 5.

Table 5
Computing environment and specifications.

Parameters	Google Colaboratory Free	Apple Macbook Pro 2017
CPU	Intel(R) Xeon(R)	Intel(R) Core(TM) i7
No. of CPU Cores	2	4
CPU Frequency	2.3 GHz	2.8 GHz
RAM	12 GB	16 GB
GPU	Nvidia K80 / T4	Radeon Pro-555
GPU Memory	12 GB	2 GB
Python version	3.7	3.8

We perform two types of experiments using the proposed methodologies: (1) traditional classification (2) unknown-class classification. Traditional classification is when a model is trained and then evaluated on all classes that were present during the training of the model. Typically, this is implemented using a Softmax-activated layer which has a fixed number of units and can only predict the class probabilities for that fixed number of classes. We perform the traditional classification experiments to show the potential of the GEM images in a standard evaluation setting and for the sake of completeness and comparison with other state of art works. Unknown-class classification is when the model is evaluated on all classes that were present during training as well as new classes that were unknown to the model during training. We implement this by withholding classes from the training dataset and use few-shot classification techniques to perform unknown-class classification.

4.1. Traditional classification experiments

For all the experiments under this section, we used the Shallow-CNN classifier. Also, all the experiments performed had the data split as 70% of the dataset for training and 30% of the dataset for evaluation. We started experimenting with various feature extraction and visualizing techniques mentioned in Section 3.2 and more. The results of these experiments are summarized in Fig. 9. We can observe that different visualization techniques result in varying performance across all three datasets. It is notable that the GEM image performed the best across all three datasets, where as other methods performed at a high accuracy for some datasets but gave worse performance on more recent datasets. These results supported our decision of using the GEM images for further experiments.

As the classification results of the GEM images are the best and most interesting, we summarize the results of traditional classification with GEM images in Fig. 11. We can observe that, this approach was able to classify 19 families with 100% accuracy, and the remaining with an accuracy higher than 98% for Dataset-A. It achieved an accuracy higher than 96% accuracy for 8 families, and approximately 64% accuracy for 1 family for the Dataset-B. It classified 4 families with a 100% accuracy, 7 families with an accuracy higher than 97% and the remaining with an accuracy higher than 90% for Dataset-C.

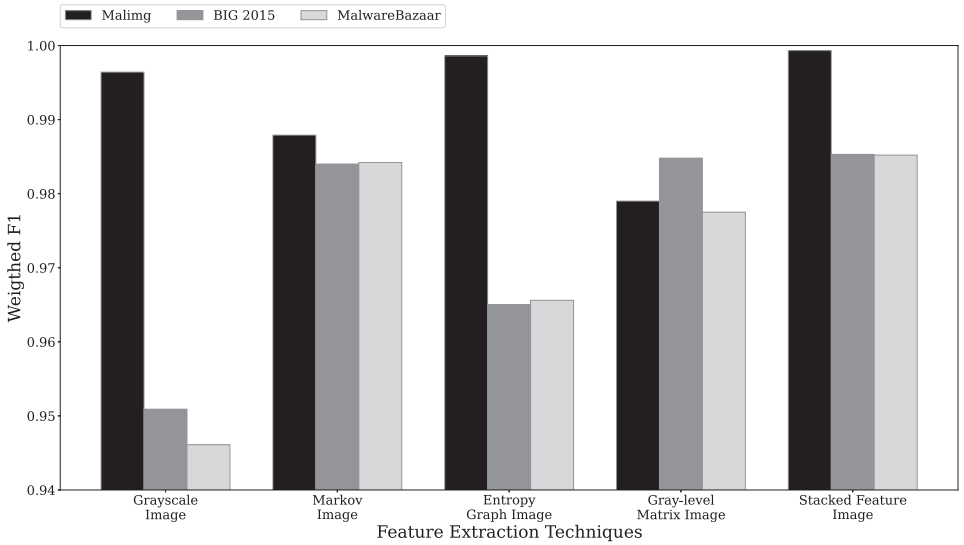


Fig. 9. Performance comparison between various feature extraction techniques. The GEM image consistently shows better performance over the grayscale and individual feature images for all three datasets. These experiments were performed using traditional classification using Shallow-CNN model.

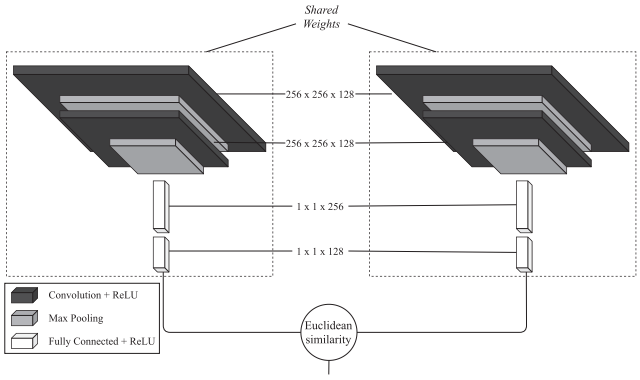


Fig. 10. The architecture of CSNN built using the feature extractor from the Shallow-CNN model. The feature extractor with the shared weights is joined using the Euclidean similarity metric.

4.2. Experiments using the CSNN model

This section describes the classification experiments we performed on the three datasets using the CSNN model. As we have mentioned previously, CSNN performs better when there are equal instances from each class, and when only a small percentage of the total data is in the training set. For all the experiments with the CSNN, we split the data such that percentage of total data in the training set was approximately 10% and the rest in the evaluation set. In all further experiments, we trained the CSNN using the the Adam optimizer with a learning rate of *0.0001*. We used the Early Stopping regularization technique to efficiently train the CSNN as well as to avoid overfitting. Early stopping usually caused the training of CSNN to stop after 10–12 epochs irrespective of the dataset. Typically, the CSNN model is used in 1-shot classification scenarios, and thus, we also experiment on CSNN with 1-shot classification.

4.2.1. 1-Shot traditional classification experiments

For these experiments, the CSNN is trained and evaluated on all classes present in the dataset. We evaluate the CSNN using 1-shot classification, implying that 1 instance from each class is selected to be in the support set at random. The training set is formed by selecting a certain number of samples from each class at random. This number is decided such that the number of samples in the training set are approximately 10% of the total data. Table 6 describes the training set formation for each dataset in greater detail. Dataset-B contains a family with only 42 samples, but we require 150 samples from each class to have an equal number of instances from each class. In practice, we would tackle this problem by withholding this class from the training set. For the sake of completeness, we use the random oversampling technique which involves selecting, with replacement, instances from a minority class at random. We ensured that no samples from the training set overlapped with the evaluation set.

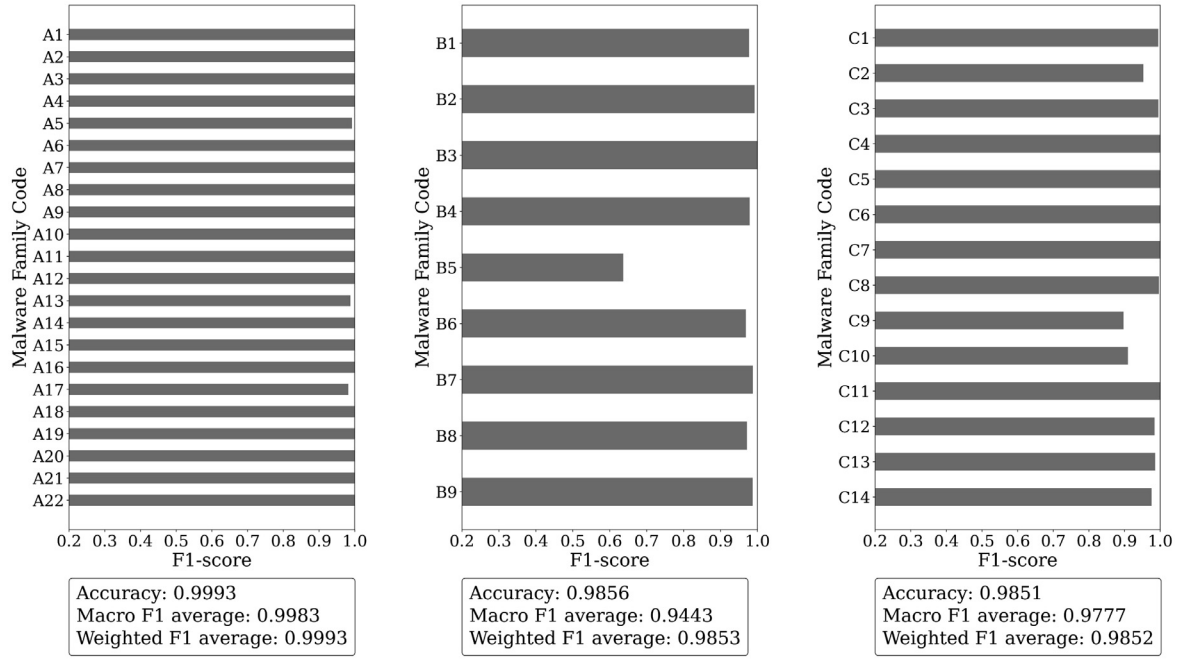
Table 6
Training set details for Experiment-1.1: Few-shot traditional classification with CSNN.

Dataset	No. of classes in training set	No. of samples selected per class	Total no. of samples in training set	% of total data
Dataset-A	22	50	1100	11.77%
Dataset-B	9	150	1350	12.43%
Dataset-C	14	60	840	10.39%

The results of the experiments (refer Fig. 12) are competitive with the state of art results for traditional classification, considering that the CSNN was trained on 10% of the dataset. The 1-shot traditional classification with CSNN was able to classify 11 families with 100% accuracy, 8 families with an accuracy higher than 99%, and the remaining with an accuracy higher than 86% for Dataset-A. It classified 6 families with an accuracy higher than 95% and the remaining families with an approximately 85% accuracy for Dataset-B. It classified 9 families with an accuracy higher than 97%, 3 families with an accuracy higher than 94%, and the remaining with an accuracy higher than 77%.

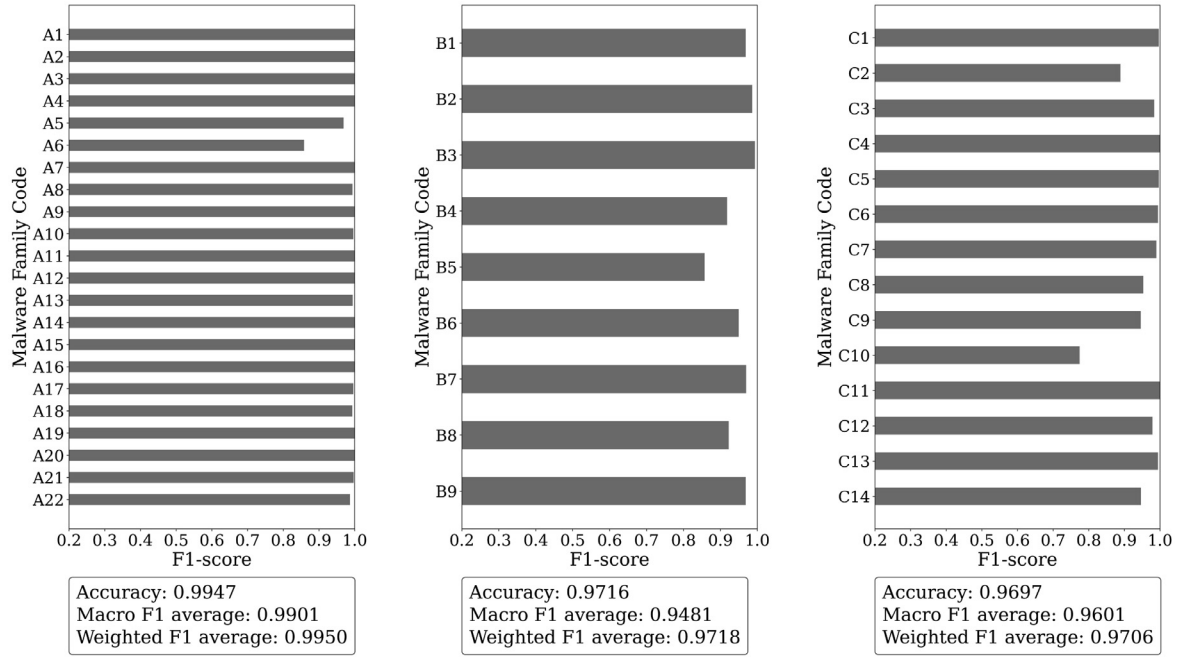
4.2.2. 1-Shot unknown-class classification

To emulate unknown-family recognition similar to a real world setting, we withheld classes from the training set and these are only presented to the CSNN while evaluation. The number of classes withheld in the training set is proportional to the total number of classes in the dataset. For Dataset-A, Dataset-B and Dataset-C, the number of classes withheld are 5, 2 and 3 respec-



(a) Malimg dataset (b) Microsoft BIG 2015 (c) Malbaz dataset

Fig. 11. Results for the traditional classification using our Shallow-CNN model.



(a) Malimg dataset (b) Microsoft BIG 2015 (c) Malbaz dataset

Fig. 12. Results for the 1-shot traditional classification using our CSNN model.

tively. The classes with the least number of instances are selected to be withheld. The splitting of data into the training set and testing set is similar to the previous experiment. Table 7 describes the exact split of the data. One sample from each class in the testing set was selected to be in the support set at random. The results of the experiments (refer Fig. 13) outperform state of art results for unknown-class classification using CSNN.

4.3. Experiments using the Shallow-FS model

This section describes the classification experiments we performed on the three datasets using the Shallow-FS model. For these experiments, we split the data into the training set and the testing set. 70% of data from each class forms the training set, and the remaining 30% of the data forms the testing set. The Shallow-

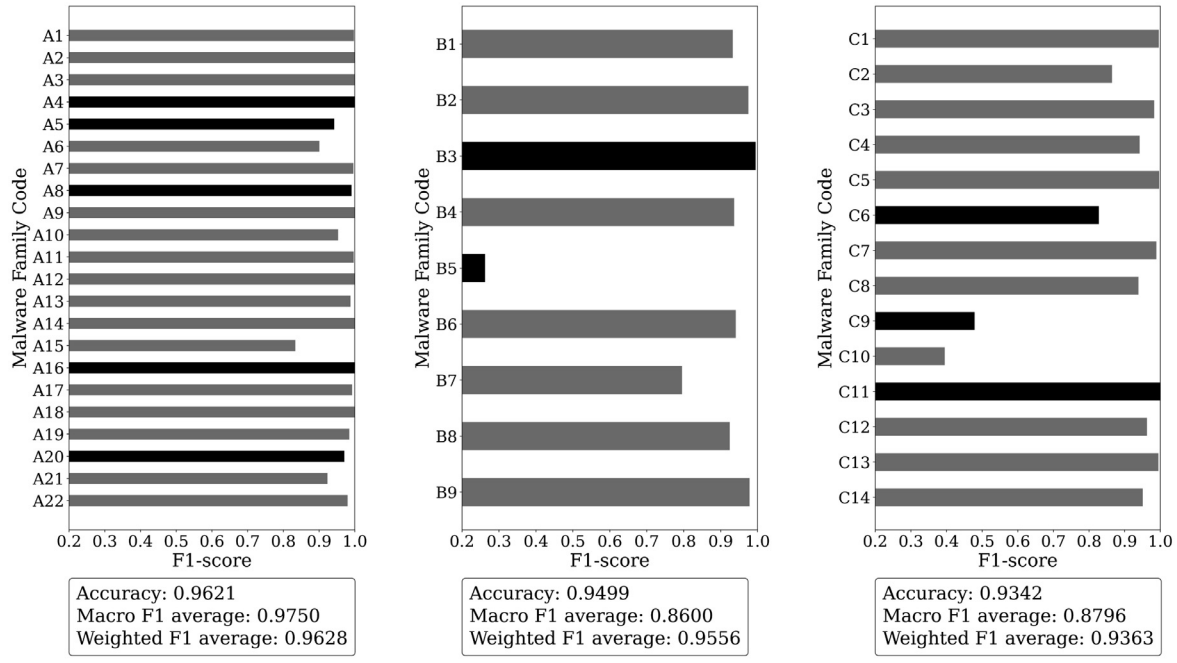


Fig. 13. Results for the 1-shot unknown-class classification using our CSNN model. The classes in black represent classes withheld from the training set and only present in the evaluation set. The classes in gray are present in both the sets.

Table 7
Training set details for Experiment-1.2: One-shot novel-class classification with CSNN.

Dataset	No. of. classes in training set	No. of samples selected per class	Total no. of samples in training set	% of total data
Malimg	17	50	850	9.10%
BIG 2015	7	150	1050	9.66%
MalBaz	11	60	660	8.17%

FS was trained using the Adam optimizer with a learning rate of *0.0001*. We used the Early stopping regularization technique to efficiently train the Shallow-FS and to avoid overfitting. Early stopping usually caused the training of Shallow-FS to stop after 4–6 epochs irrespective of the dataset. We already performed 1-shot classification experiments using the CSNN model, so we decided to experiment with 10-shot classification using the Shallow-FS model.

4.3.1. 10-Shot traditional classification

For these experiments, the Shallow-FS is trained and evaluated on all classes present in the dataset. We evaluate the Shallow-FS using 10-shot classification, implying that 10 instances from each class are selected to be in the support set at random. Furthermore, the feature vectors of the 10 selected instances from each class were averaged to form the mean feature vectors. These mean feature vectors are then provided, in combination with the feature embedding of the query instance, to the Cosine similarity metric.

The results of the experiments (refer Fig. 14) are competitive with the state of art traditional classification results. Comparing it to the Shallow-CNN experiments, as both these experiments had similar split of the data, we observe the Shallow-FS performs slightly worse for Dataset-B and Dataset-C. The Shallow-FS was able to classify 19 families with a 100% accuracy, and the remaining with an accuracy higher than 98% for Dataset-A. It achieved an accuracy higher than 97% for 7 families, an accuracy higher than 94% for 1 family and 74% accuracy for the remaining family for Dataset-B. It classified 3 families with 100% accuracy, 7 families with an accuracy higher than 97% and the remaining with an accuracy higher than 81%.

4.3.2. 10-Shot unknown-class classification

To emulate unknown-family recognition similar to a real world setting, we withheld classes from the training set. The withholding of classes is exactly the same as 1-shot unknown-class classification using CSNN (refer Section 4.2.2). For this experiment, we select ten images from each class to be in the support set at random. The results of the experiments (refer Fig. 15) outperform state of art results for unknown-class classification using few-shot classification. Interestingly, when comparing the Shallow-FS with the CSNN, we can observe that the Shallow-FS is better at classifying the unknown classes (withheld classes). While the CSNN struggles to successfully classify a few of the unknown-classes, the Shallow-FS can classify most with high accuracy.

Fig. 15 clearly demonstrates that this approach provides the best results for unknown class classification. In order to stress test this classifier, we extended Dataset-C with two variants of the Linux-based **Mirai** malware. We collected 200 Mirai files from MalwareBazaar public malware repository (2021) in June of 2022 and separated them into two classes, which were unobfuscated Mirai and UPX-obfuscated Mirai. We conducted this exact experiment on the extended Dataset-C. The overall accuracy and weighted F1-score were 0.9380 and 0.9367 respectively. From these results, we can infer that shallow-FS can classify recently distributed malware. We also show that the GEM image technique is independent of malware platform, which implies that it can classify malware intended for any operating system as well as memory dumps.

5. Discussion

In this section, we discuss the results of the experiments mentioned in the previous section in greater detail. We summarize a

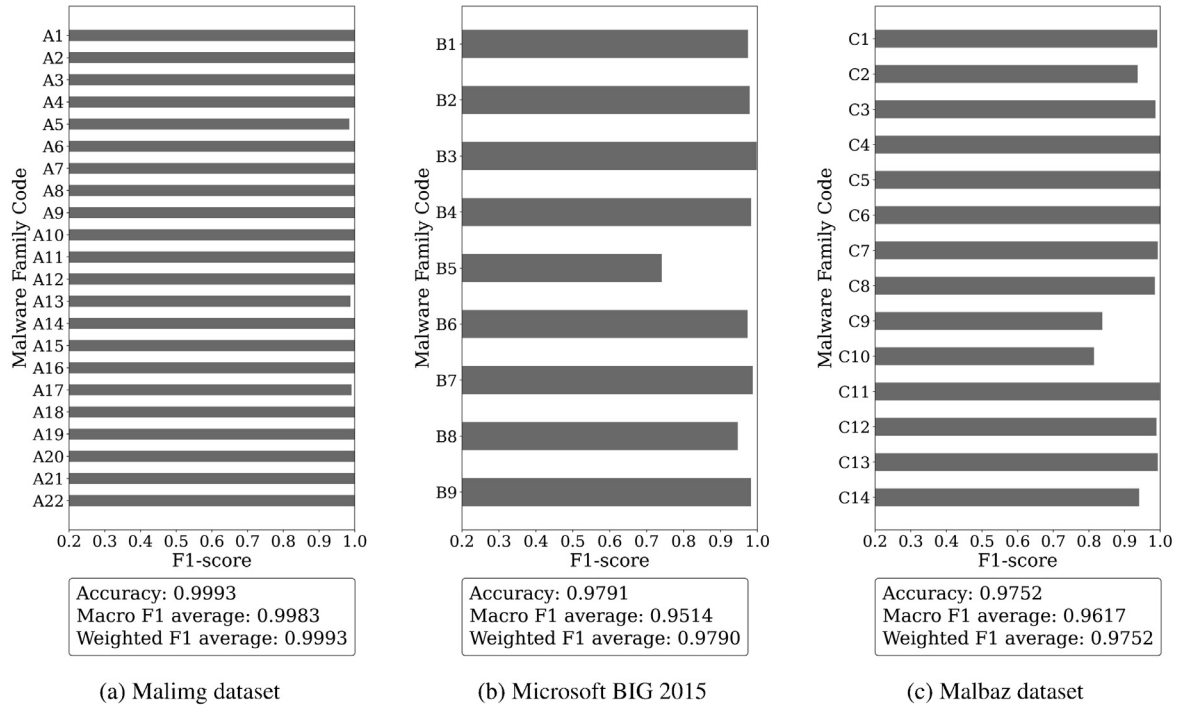


Fig. 14. Results for the 10-shot traditional classification using Shallow-FS.

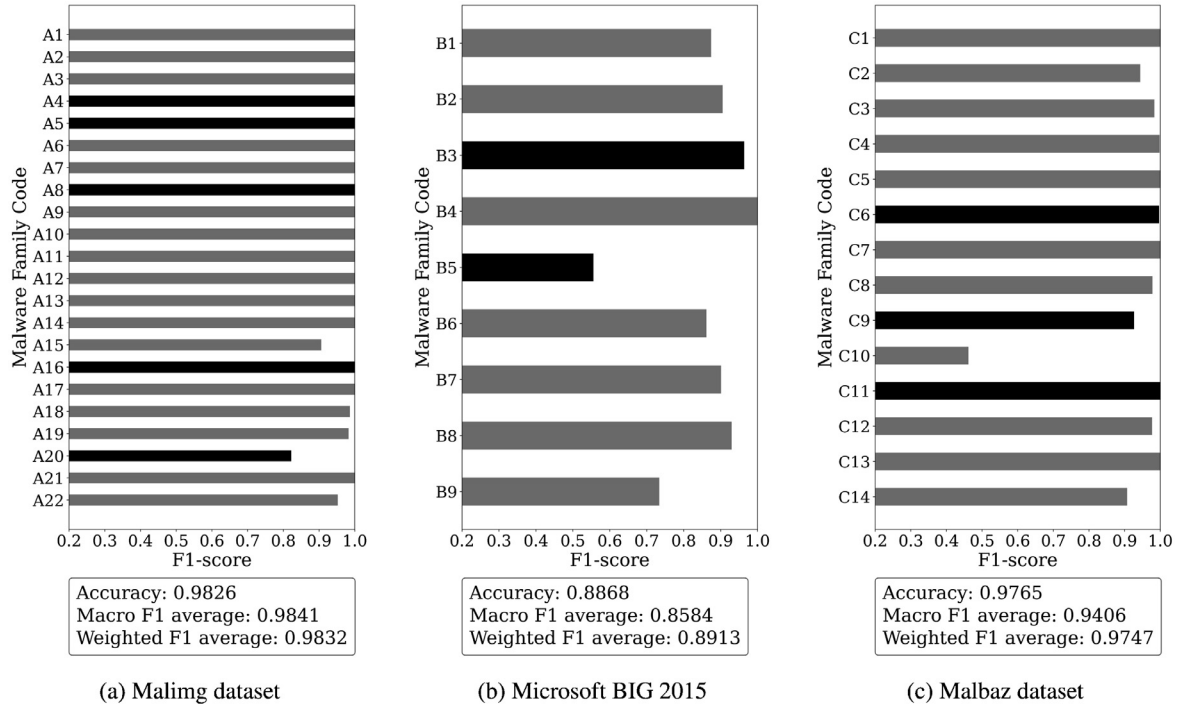
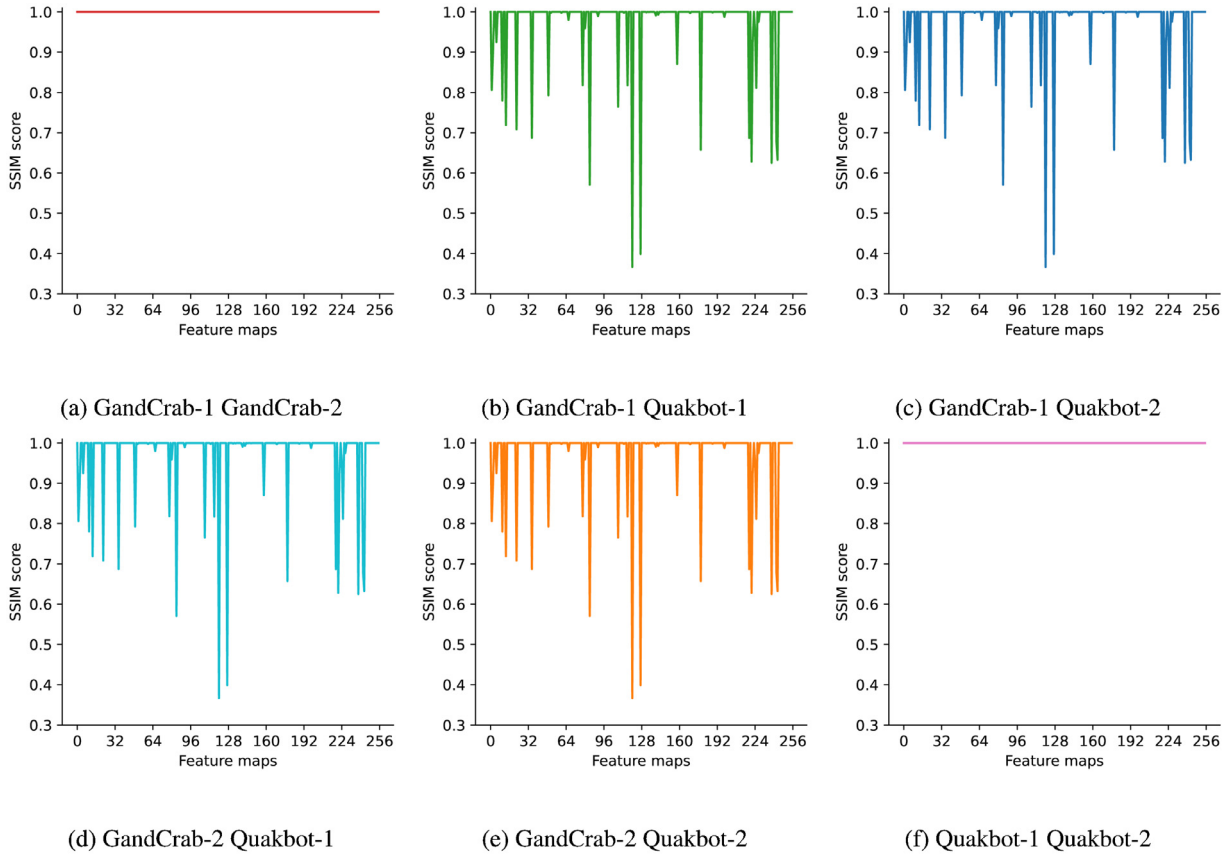


Fig. 15. Results for the 10-shot unknown-class classification using Shallow-FS. The classes in black represent classes withheld from the training set and only present in the evaluation set. The classes in gray are present in both the sets.

few general misclassification trends which were observed in the experiment results. It is not possible to provide concrete reasoning for the misclassification. A common trend that we observed across all classification models was for the samples of family C10 (RedLineStealer) being misclassified as family C9 (CryptBot). We believe that the reason behind the misclassification between these two families is because of the close similarities in their behaviour (Various Types of Threats, 2021). It also possible that, a few samples on MalwareBazaar may be mislabelled as it is a public repository. We also found a few remarks on the Internet about both the malware families sharing some codebase and dropping mechanisms.

Another interesting result we observed was the CSNN outperforming the Shallow-FS model for Dataset-B. CSNN achieved an accuracy of 94.99%, where as Shallow-FS achieved an accuracy of 88.68%. On closer observation, we can note that the Shallow-FS



(a) GandCrab-1 GandCrab-2 (b) GandCrab-1 Quakbot-1 (c) GandCrab-1 Quakbot-2

(d) GandCrab-2 Quakbot-1 (e) GandCrab-2 Quakbot-2 (f) Quakbot-1 Quakbot-2

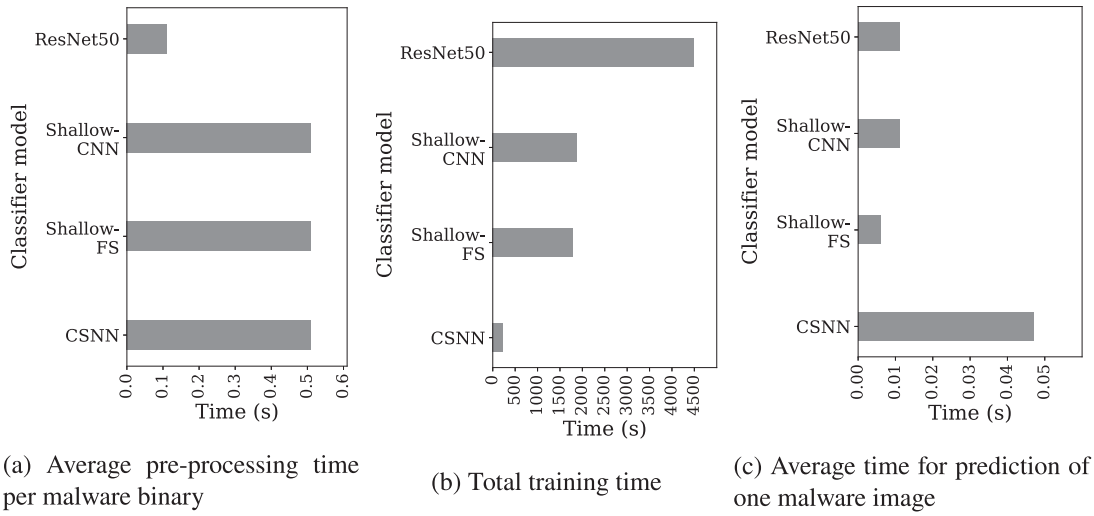
Fig. 16. SSIM analysis of feature embeddings.

classified the unknown and complex to classify family such as B5 better than CSNN. A few other families had many variations among the instances of the family. When we selected ten random instances, for the support set of the 10-shot classification experiment, the support set feature vectors would have a very high variance. On the other hand, support set feature vectors for 1-shot classification with the CSNN would not have such high variance as the 10-shot classification. Thus we believe, that the noise introduced in the 10-shot support set feature vector caused Shallow-FS to perform worse than the CSNN. It is important to note that, generally, Dataset-B has more noise because of the missing bytes in the malware samples represented by ''??''. This resulted in an overall low performance for all our approaches. We also did not use the *.asm* file provided in the Dataset-B as our goal was to keep the pre-processing of the raw binary as minimal as possible.

The shallow CNN architecture we propose the use of in our work are far from the norm in state of art malware visualization based research works. We would prefer to not use our model blindly as a black-box. Thus, we take the Shallow-CNN model, pre-trained on Dataset-C, and remove all the fully-connected layers. As a result, we get a feature extractor that outputs the result of the feature extraction process. The output is an image of shape $62 \times 62 \times 256$, which contains 256 feature maps of size 62×62. We take two instances from two distinct classes of Dataset-C, namely Quakbot and GandCrab, and provide them as input to the feature extractor in all possible permutations (refer Fig. 16). It outputs 6 different pairs of feature vectors which are recorded. We then use the Structural Similarity Index Measure (SSIM) to calculate the similarities between all the feature maps of all 6 pairs. We can clearly observe that the features extracted between the instances of the same class have a consistently high SSIM score across all feature maps. On the other hand, instances of different classes show a lot of variance in the SSIM score across all feature maps.

Previously, we make comparison of the GEM image fusion technique with the individual feature extraction techniques. We observed that the GEM image technique performed the best out of them all. For the sake of completeness, we ran four additional experiments using the Shallow-CNN and the state of art Transfer learning model ResNet50. We compare the two different feature extraction techniques, grayscale image colour-mapped and the GEM image, using the two models. Fig. 18 summarizes the results of these experiments. We can clearly observe that the GEM image outperforms the other technique with both the models. For both the models, the GEM image showed better performance than the grayscale colour-mapping technique used by various works (Dai et al., 2018; Vasan et al., 2020a; 2020b). This also shows the compatibility of our GEM image technique with the state of the art models for better performance over the traditional visualization techniques.

As we mentioned previously, due to the COVID-19 pandemic, we had access to very limited hardware. All of our experiments were performed on Google Colaboratory, which provides no guarantees for resource availability. Thus the time comparison of our different models is not precisely accurate, but it still provides a good approximation on the time taken by each of the classification models. We also provide the comparison of our models with state of art Transfer learning models such as ResNet in Fig. 17. For the pre-processing of input for ResNet, we do not perform any data augmentation and thus, do not take the time taken by any data augmentation techniques into account. As observed, the pre-processing time for our approach is much higher because of the generation of the GEM image as compared to the standard pre-processing of ResNet. The training time of the CSNN is the low-



(a) Average pre-processing time per malware binary

(b) Total training time

(c) Average time for prediction of one malware image

Fig. 17. Time based analysis performed for all our classifier models and ResNet50. We recorded and plotted the time taken for pre-processing, training and evaluation/prediction tasks.

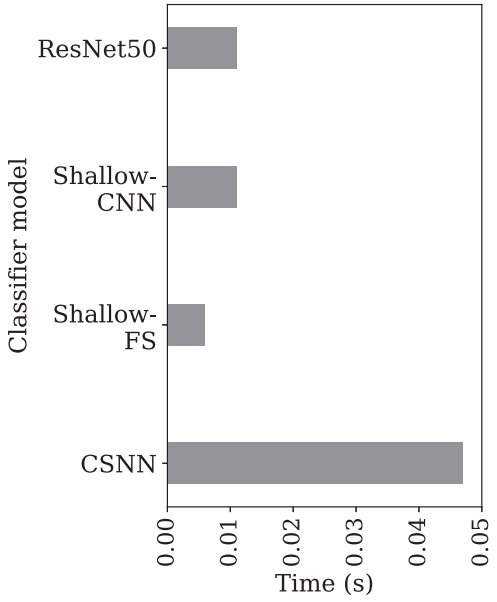


Fig. 18. A bar plot showing the difference in performance for the ResNet50 and the Shallow-CNN architectures. Our model outperforms ResNet50 in both coloured-image malware visualization techniques, namely colour-mapped grayscale images and the GEM images. This shows the potential of the GEM image and its compatibility with the state of the art transfer learning models over colour-mapped grayscale images.

est because of the less training data. The rest of our models take less time to train than the ResNet because of the shallower architecture. Interestingly, Shallow-FS model takes the least amount of time for prediction. We can observe, that on average, using a Cosine similarity score and support set to classify instead of a softmax classifier is faster for classification in our experimental setting. The reason behind the higher average prediction time of CSNN is that it processes two images at a time. We can reduce this time by extracting the shared CNN in the CSNN and generating support set feature vectors and query image feature vectors separately. We store the values of the support set feature vectors instead of recalculating them for every prediction.

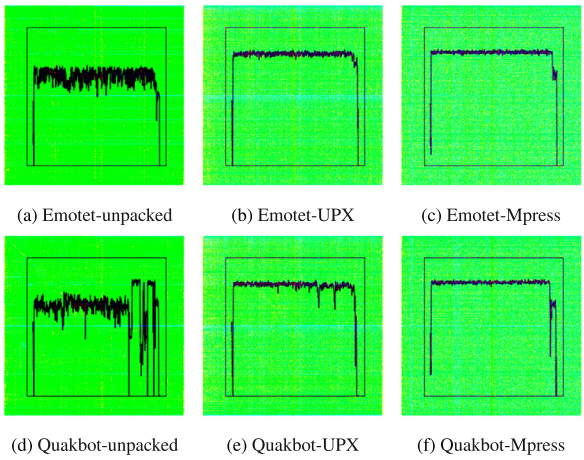
The summary of our results and comparison with the state of the art research works is presented in Table 8. The rows in bold

Table 8
Performance comparison of our approaches with the state-of-the-art research.

Authors	Year	Classification Type	Malimg accuracy (%)	Microsoft BIG 2015 accuracy (%)	Malbaz accuracy (%)
Nataraj et al.	2011	Traditional	98.08	-	-
Ni et al.	2018	Traditional	-	99.26	-
Vasan et al.	2020	Traditional	98.82	-	-
Xiao et al.	2020	Traditional	99.70	100	-
Gibert et al.	2020a	Traditional	-	99.75	-
Vasan et al.	2020b	Traditional	99.50	-	-
Shallow-CNN	**-**	**Traditional**	**99.93**	**98.56**	**98.51**
CSNN	**-**	**Traditional**	**99.47**	**97.16**	**96.97**
Shallow-FS	**-**	**Traditional**	**99.93**	**97.91**	**97.52**
Tran et al.	2019	Unknown-class	95.30	70.19	-
CSNN	**-**	**Unknown-class**	**96.21**	**94.99**	**93.42**
Shallow-FS	**-**	**Unknown-class**	**98.26**	**88.68**	**97.65**

represent our proposed models' performance. None of the prior research works listed in Table 8 discuss few-shot learning on PE malware except for (Tran et al., 2019). The results from the research works (Gibert et al., 2020a; Ni et al., 2018; Xiao et al., 2020) are marginally better than our classifier models for Dataset-B. The paper by Ni et al. (2018) uses the disassembled *.asm* files provided for each malware family to conduct their classification and their pre-processing technique removed approximately 60 files from the dataset. Relying on disassemblers could make the classifier sensitive to sophisticated obfuscation and add to the pre-processing time of a malware binary. Also, the accuracy of 99.26% is their best case accuracy where as their average case accuracy was 98.86%. The method proposed by Gibert et al. (2020a) also relies on the disassembled files to produce an accuracy of 99.75%. Their bytes-based classifier, which only considers the *.bytes* files achieved an accuracy of 97.56%. To the best of our knowledge, the method of Xiao et al. (2020) performed the evaluation on the actual evaluation set provided in the Microsoft BIG 2015 Kaggle challenge (refer Section 3.1.2) which is not publicly available. The comparison between these state of the art works and our work is not entirely fair as our methodologies and evaluation settings differ conceptually.

Finally, we discuss the impact of obfuscation on our GEM static feature extraction technique. Dataset-A and Dataset-B had the Obfuscator.AD (family code A16) and the Obfuscator.ACY (family code B8) family respectively (refer Tables 1, 2). All families in Dataset-C



(a) Emotet-unpacked (b) Emotet-UPX (c) Emotet-Mpress

(d) Quakbot-unpacked (e) Quakbot-UPX (f) Quakbot-Mpress

Fig. 19. Structural differences observed between GEM images of an unobfuscated executable of Emotet and Quakbot family. Structural similarities observed between GEM images of the UPX and Mpress packed executables of the same families.

used obfuscation which we verified using third-party vendors on MalwareBazaar public malware repository (2021) (refer Table 3). We have shown that our classifiers are able to classify the executables in these obfuscated families with a high F1-score (refer Figs. 11–15). We achieved good results for Dataset-C because we observed that each malware family in the dataset uses its own obfuscation techniques/packers. This phenomenon is often observed in the wild with malware families using custom obfuscation techniques to evade signature-based detection. To better analyze the GEM image construction on obfuscated files we picked one executables from the Quakbot (family code C7) and Emotet (family code C12) families and obfuscated them with commonly-known packers such as UPX: the Ultimate Packer for eXecutables (2021) and Mpress executable packer (2021). We manually unpacked the executables before performing the analysis. Fig. 19 highlights the structural similarities between the executables from two distinct families caused by obfuscation. We can observe that the GEM images of the unpacked malware are distinct enough to be classified accurately however, problems would arise when two distinct malware families use the same obfuscation technique or tool. It is highly probable that the similarities in this image will cause a misclassification. This worst-case scenario can be tackled using the GEM image technique on memory dumps extracted by dynamic analysis of the obfuscated binaries instead of the static binary content.

6. Conclusion

This research proposed a the GEM image that is more compatible with shallower CNN architectures than the traditional deeper architectures, allowing for quicker training and classification. We proposed a Gray-level matrix image that enabled us to visualize GLCM based textural features combined with the Markov image and the Entropy graph image. Fusing the three feature extraction techniques allowed us to visualize malware in a novel way that is still compatible with state of the art CNN architectures. We experimentally show that the GEM image format combined with a shallow CNN architecture showed competitive results for the traditional classification and better results than the state-of-the-art few-shot malware classification research. Our proposed CNN architectures can be used to tackle malware-specific problems such as the scarcity of samples for specific families and the need to classify unknown families because of the evolution of malware families and zero-day attacks. We experimentally show that our shallow architecture, Shallow-CNN, performs better when combined with the GEM feature extraction and fusion technique. Our few-shot classification are still compatible with the traditional malware classification, and can further be used to classify unknown malware families. We provide reasons for some of the classifications our classifier models make aiming towards improved performance in future works. Inspired by Ayyar et al. (2021), we also try to analyze the intermediate feature map images of our models which provides us a better understanding of our model. Lastly, we compare our work to the state of the art and show the performance improvement we achieved.

For future work, our approach can be applied to dynamic malware analysis, by generating GEM images of memory dumps instead of raw malware binaries. This would allow the classifier to be more robust against sophisticated obfuscation techniques. The feature extraction techniques we use make our approach easily applicable to classifying Internet of Things (IoT) malware, Unix-based malware, Mobile malware etc. as it does not rely on specifics of a malware binary. Our approach focuses on fusing the 3 distinct feature images to form the GEM image, but selection of other feature extraction techniques that do not result in a single feature image can be amalgamated via an ensemble classifier.

Declaration of Competing Interest

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

CRediT authorship contribution statement

Mauro Conti: Conceptualization, Methodology, Validation, Investigation, Writing – review & editing, Supervision. **Shubham Khandhar:** Conceptualization, Software, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Supervision. **P. Vinod:** Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Supervision.

References

Alrabaee, S., Shirani, P., Wang, L., Debbabi, M., 2018. FOSSIL: a resilient and efficient system for identifying foss functions in malware binaries. ACM Trans. Privacy Secur. (TOPS) 21 (2), 1–34.

AV Test malware statistics, 2021. AV Test malware statistics. https://www.av-test.org/en/statistics/malware(accessed 20 October 2021).

Ayyar, M. P., Benois-Pineau, J., Zemmari, A., 2021. White box methods for explanations of convolutional neural networks in image classification tasks. arXiv preprint arXiv:2104.02548.

Bai, Y., Xing, Z., Li, X., Feng, Z., Ma, D., 2020. Unsuccessful story about few shot malware family classification and siamese network to the rescue. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, pp. 1560–1571.

Bromley, J., Bentz, J.W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Säckinger, E., Shah, R., 1993. Signature verification using a æsiameseg time delay neural network. Int. J. Pattern Recognit. Artif. Intell. 7 (04), 669–688.

Burguera, I., Zurutuza, U., Nadjm-Tehrani, S., 2011. Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15–26.

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., Huang, J.-B., 2019. A closer look at few-shot classification. arXiv preprint arXiv:1904.04232.

Chuang, H.-Y., Wang, S.-D., 2015. Machine learning based hybrid behavior models for android malware analysis. In: 2015 IEEE International Conference on Software Quality, Reliability and Security. IEEE, pp. 201–206.

Dai, Y., Li, H., Qian, Y., Lu, X., 2018. A malware classification method based on memory dump grayscale image. Digital Invest. 27, 30–37.

Fei-Fei, L., Fergus, R., Perona, P., 2006. One-shot learning of object categories. IEEE Trans. Pattern Anal. Mach. Intell. 28 (4), 594–611.

Gibert, D., Mateu, C., Planes, J., 2020. HYDRA: a multimodal deep learning framework for malware classification. Comput. Secur. 95, 101873.

Gibert, D., Mateu, C., Planes, J., 2020. The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. 153, 102526.
Google Colaboratory, 2021. Google Colaboratory. https://colab.research.google.com/signup(accessed 20 October 2021).
Han, K.S., Lim, J.H., Kang, B., Im, E.G., 2015. Malware analysis using visualized images and entropy graphs. Int. J. Inf. Secur. 14 (1), 1–14.
Hsiao, S.-C., Kao, D.-Y., Liu, Z.-Y., Tso, R., 2019. Malware image classification using one-shot learning with siamese networks. Procedia Comput. Sci. 159, 1863–1871.
IT threat evolution, 2021. IT threat evolution Q2 2021. https://securelist.com/it-threat-evolution-in-q2-2021-pc-statistics/103607/(accessed 20 October 2021).
Khandhar, S., 2021. A few-shot malware classification approach for unknown family recognition using malware feature visualization.
Ki, Y., Kim, E., Kim, H.K., 2015. A novel approach to detect malware based on API call sequence analysis. Int. J. Distrib. Sens. Netw. 11 (6), 659101.
Koch, G., Zemel, R., Salakhutdinov, R., et al., 2015. Siamese neural networks for one-shot image recognition. ICML Deep Learning Workshop, Vol. 2. Lille.
LeCun, Y., Bengio, Y., et al., 1995. Convolutional networks for images, speech, and time series. Handb. Brain Theory Neural Netw. 3361 (10), 1995.
Makandar, A., Patrot, A., 2017. Malware class recognition using image processing techniques. In: 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI). IEEE, pp. 76–80.
MalwareBazaar public malware repository, 2021. MalwareBazaar public malware repository. https://bazaar.abuse.ch/(accessed 20 October 2021).
Mcafee ATR Threat Reports, 2021. Mcafee ATR Threat Reports April 2021. https://www.mcafee.com/enterprise/en-us/lp/threats-reports/apr-2021.html(accessed 20 October 2021).
Microsoft Threat report, 2009. Microsoft Threat report Win32/C2Lop.gen!L. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/C2Lop.gen!L(accessed 20 October 2021).
Microsoft Threat report, 2010. Microsoft Threat report Win32/C2Lop.gen!M. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/C2Lop.gen!M(accessed 20 October 2021).
Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
Mpress executable packer, 2021. Mpress executable packer. https://www.autohotkey.com/mpress/mpress_web.htm(accessed 3 July 2022).
Natani, P., Vidyarthi, D., 2013. Malware detection using API function frequency with ensemble based classifier. In: International Symposium on Security in Computing and Communication. Springer, pp. 378–388.
Nataraj, L., Jacob, G., Manjunath, B., 2010. Detecting Packed Executables based on Raw Binary Data. VRL, ECE.
Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S., 2011. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 1–7.
Ni, S., Qian, Q., Zhang, R., 2018. Malware identification using visualization images and deep learning. Comput. Secur. 77, 871–885.
Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A., 2015. Malware classification with recurrent networks. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1916–1920. doi:10.1109/ICASSP.2015.7178304.
Qiao, Y., Jiang, Q., Jiang, Z., Gu, L., 2019. A multi-channel visualization method for malware classification based on deep learning. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 757–762. doi:10.1109/TrustCom/BigDataSE.2019.00109.
Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M., 2021. Microsoft malware classification challenge. abs/1802.10135.
Rong, C., Gou, G., Hou, C., Li, Z., Xiong, G., Guo, L., 2021. UMVD-FSL: unseen malware variants detection using few-shot learning. In: 2021 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1–8.
SANS Webcast Recap, 2020. SANS Webcast Recap 2020. https://www.vmray.com/cyber-security-blog/practical-malware-family-identification-sans-webcast-recap/(accessed 20 October 2021).
Segaran, T., 2007. Collective Intelligence-Building Smart Web 2.0 Applications. Newton: O'Reilly.
Tran, T.K., Sato, H., Kubo, M., 2019. Image-based unknown malware classification with few-shot learning models. In: 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW). IEEE, pp. 401–407.
UPX: the Ultimate Packer for eXecutables, 2021. UPX: the Ultimate Packer for eXecutables. https://upx.github.io/(accessed 3 July 2022).
Various Types of Threats, 2021. Various Types of Threats Disguised as Software Download Being Distributed. https://asec.ahnlab.com/en/26274/(accessed 20 October 2021).
Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q., 2020. IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Netw. 171, 107138.
Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q., 2020. Image-based malware classification using ensemble of CNN architectures (IMCEC). Comput. Secur. 92, 101748.
Vu, D.-L., Nguyen, T.-K., Nguyen, T.V., Nguyen, T.N., Massacci, F., Phung, P.H., 2020. HIT4Mal: hybrid image transformation for malware classification. Trans. Emerg. Telecommun.Technol. 31 (11), e3789.
Wang, P., Tang, Z., Wang, J., 2021. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. Comput. Secur. 106, 102273.
Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M., 2020. Generalizing from a few examples: a survey on few-shot learning. ACM Comput. Surv. (CSUR) 53 (3), 1–34.
Xiao, G., Li, J., Chen, Y., Li, K., 2020. MalFCS: an effective malware classification framework with automated feature extraction based on deep convolutional neural networks. J. Parallel Distrib. Comput. 141, 49–58.
Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S., 2017. A survey on malware detection using data mining techniques. ACM Comput. Surv. (CSUR) 50 (3), 1–40.
Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X., 2020. Byte-level malware classification based on markov images and deep learning. Comput. Secur. 92, 101740.

Mauro Conti is Full Professor at the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his PhD, he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor at the University of Padua, where he became Associate Professor in 2015, and Full Professor in 2018. He has been Visiting Researcher at GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest is in the area of Security and Privacy. In this area, he published more than 350 papers in topmost international peer-reviewed journals and conferences. He is Area Editor-in-Chief for IEEE Communications Surveys & Tutorials, and Associate Editor for several journals, including IEEE Communications Surveys & Tutorials, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, and IEEE Transactions on Network and Service Management. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, and General Chair for SecureComm 2012, SACMAT 2013, CANS 2021, and ACNS 2022. He is Senior Member of the IEEE and ACM. He is a member of the Blockchain Expert Panel of the Italian Government. He is Fellow of the Young Academy of Europe.

Shubham Khandhar received his MSc degree in computer science with a 4TU specialisation in cybersecurity from the Delft University of Technology, Delft, Netherlands. He holds a BSc degree in computer science from the Vrije Universiteit Amsterdam, Amsterdam, Netherlands. His research area of interests include Malware Analysis, Cyber Threat Intelligence, Deep Learning and Few-shot learning.

Vinod P., is presently a Professor in the Department of Computer Applications at Cochin University of Science & Technology, Kochi, Kerala, India. He was a Postdoctoral Researcher at the Department of Mathematics, University of Padua, Italy, where he was a part of the EUH2020 project named TagitSmart. He was also a Postdoctoral researcher at Malaviya National Institute of Technology, Jaipur, Rajasthan under the ISEA project on Mobile Security. He holds his PhD in Computer Engineering from Malaviya National Institute of Technology, Jaipur, India. He is awarded Marie Skodowska-CurieFellowship (Project Title OPTIMA: Organization Specific Threat Intelligence Mining and Sharing) by the European Commission. He has numerous research articles published in peer-reviewed Journals and International Conferences. He also serves as a programme committee member in the International Conferences related to Computer and Information Security. Vinods area of interest is Adversarial Machine Learning, Malware Analysis, Context-aware privacy-preserving Data Mining, and Natural Language Processing.is