A Comprehensive Study on Learning-Based PE Malware Family Classification Methods

Yixuan Ma
State Key Laboratory of Communication Content Cognition
Tianjin, China
College of Intelligence and Computing, Tianjin University
Tianjin, China
2019216047@tju.edu.cn

Shuang Liu*
College of Intelligence and Computing, Tianjin University
Tianjin, China
shuang.liu@tju.edu.cn

Jiajun Jiang
College of Intelligence and Computing, Tianjin University
Tianjin, China
jiangjiajun@tju.edu.cn

Guanhong Chen
College of Intelligence and Computing, Tianjin University
Tianjin, China
chenguanhong@tju.edu.cn

Keqiu Li
College of Intelligence and Computing, Tianjin University
Tianjin, China
keqiu@tju.edu.cn

ABSTRACT

Driven by the high profit, Portable Executable (PE) malware has been consistently evolving in terms of both volume and sophistication. PE malware family classification has gained great attention and a large number of approaches have been proposed. With the rapid development of machine learning techniques and the exciting results they achieved on various tasks, machine learning algorithms have also gained popularity in the PE malware family classification task. Three mainstream approaches that use learning based algorithms, as categorized by the input format the methods take, are image-based, binary-based and disassembly-based approaches. Although a large number of approaches are published, there is no consistent comparisons on those approaches, especially from the practical industry adoption perspective. Moreover, there is no comparison in the scenario of concept drift, which is a fact for the malware classification task due to the fast evolving nature of malware. In this work, we conduct a thorough empirical study on learning-based PE malware classification approaches on 4 different datasets and consistent experiment settings. Based on the experiment results and an interview with our industry partners, we find that (1) there is no individual class of methods that significantly outperforms the others; (2) All classes of methods show performance degradation on concept drift (by an average F1-score of 32.23%); and (3) the prediction time and high memory consumption hinder existing approaches from being adopted for industry usage.

*Corresponding author

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473925

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

Malware Classification, Deep Learning, Concept Drift

ACM Reference Format:
Yixuan Ma, Shuang Liu, Jiajun Jiang, Guanhong Chen, and Keqiu Li. 2021. A Comprehensive Study on Learning-Based PE Malware Family Classification Methods. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3468264.3473925

1 INTRODUCTION

With the rapid development of information technology, more and more user personal information and property are transferred, stored and even shared on the Internet. Driven by the high profit, underground malware industries have been consistently growing and expanding. It is reported by AVTEST [9] that there are more than 350 thousand new malicious samples registered every day, and the total number of malware has grown roughly 11 times over the past nine years, i.e., from 99 million in 2012 to 1,139 million in 2020. Various malware families evolve fast, and tend to attack social and health events to enlarge the damage they can cause. According to the 2020 SonicWall Cyber Threat Report [55], over the past half year, attacks related to COVID-19 were increased. The biochemical systems at an Oxford university research lab currently studying the Covid-19 pandemic has been breached [5].

It is well recognized that malware family evolves fast and the concept drift, which is the change in the statistical properties of an object in unforeseen ways, has become a rather challenging issue in the domain [28]. Malware family classification is of great importance to understand the malware family evolving trend and to better detect malware [22]. This is particularly significant for portable executable (PE) malware, as Windows is one of the most widely used operating systems. According to MalwareBazaar [7], a website which offers free malware sample upload and download

services, and some existing research findings [18, 62], PE files account for around 50% of the malware uploaded by user. A large number of research approaches have been proposed to solve the PE malware classification problem, among which learning-based methods [18, 40, 62] have recently become popular benefiting from the success of machine learning, especially deep learning techniques.

Learning-based PE malware family classification techniques can mainly be categorized into three classes based on the malware file format adopted. The first class of methods convert PE malware files into images, and then apply image classification techniques to solve the problem [8, 18, 39, 40, 62]. The second class of methods directly take the PE malware file as a sequence of binary code and usually sequential models from the domain of Natural Language Processing (NLP) are adopted to solve the problem [10, 26, 43, 44, 46, 51]. More recently, a new type of methods, which de-compile the PE malware file into assembly code, and then adopt graph structure analysis on the control flow graph (CFG) of the assembly code [20, 30, 31, 65]. These three categories of methods are independently evaluated with different datasets, and they all show good performance on the evaluated dataset. The most recent approaches [62] are reported to achieve more than 98% F1-score on the BIG-15 [50] dataset.

There are some survey papers that summarize existing methods of malware family classification [17, 45, 56, 61] and Iadarola et al. [23] conduct an evaluation of image-based texture feature extraction methods. However, there is still no existing study that systematically evaluates all three classes of learning-based methods on the same evaluation settings with the same datasets. Moreover, given that those PE malware family classification approaches report excellent performance on existing public of manually crafted datasets, it is not clear (1) which class of methods are more accurate and efficient; (2) how do different kinds of methods perform in circumstances of concept drift, which is a common and realistic problem for the fast evolving malware family; and more importantly, (3) what is the status of industry adoption of those methods and the future research directions to support industry requirements.

In this work, we seek to answer the questions with a well designed empirical study. We first briefly review existing approaches and select 9 most representative approaches from the three categories: 4 image-based approaches (VGG-16 [62], ResNet-50 [11, 48, 54, 62], Inception-V3 [62], IMCFN [62]), 2 binary-based approaches (CBOW+MLP [43], MalConv [32, 44]) and 3 disassembly-based approaches (MAGIC [65], Word2Vec+KNN [10, 13], MCSC [34, 41]).

Our results suggest that (1) no individual class of methods outperforms the others and the binary-based method CBOW+MLP [43] achieves the best performance across different datasets; (2) All classes of methods show performance degradation, by an average of 32.23%, in the circumstance of concept drift and CBOW+MLP [43] shows the sharpest performance degradation (by 69.62%); (3) Industry mainly adopts sandbox and pattern database to detect malware families currently, and the long prediction time, fragility to malware evolution and the high resource consumption hinders the current learning-based methods from being applied to industry practices.

The main contributions of this work are as follows:

- We conducted the first large-scale and systematic empirical study on learning-based PE malware family classification methods.
- We create two new PE malware family classification datasets, one for the normal classification purpose and one for the concept drift purpose, and we will make them public.
- We are the first to conduct evaluations on the concept drift situation with a large number of representative methods.
- We provide practical suggestions and future research directions to promote applicable research in the area of malware family classification.

All experimental data and results are publicly available to encourage future research in the community: **https://github.com/MHunt-er/Benchmarking-Malware-Family-Classification**.

2 RELATED WORK

2.1 Image-Based Techniques

Image-based Techniques first transform the malware binary files into either grayscale or color images, and then adopt image classification models for malware family classification.

Nataraj et al. [40] propose to transform malware binary files into grayscale images, and employ K-Nearest Neighbor (KNN) model for malware classification via leveraging the texture features of images. This is also the first work that visualizes malware files into images for classification purposes. Since then, many approaches have been proposed to further improve it. For example, Ahmadi et al. [8] adopt boosting tree classifiers to better capture multiple novel features, such as Haralick and Local Binary Patterns (LBP). Narayanan et al. [39] study the performance of various machine learning models with texture features extracted via Principal Component Analysis. However, those methods are typically inefficient due to the high overhead of extracting complex texture features.

With the recent advances of deep learning models in image classification tasks, they also attract attention in the security community, and have been employed for malware family classification. For example, Convolutional Neural Networks (CNN) have been widely adopted by many approaches [18, 25, 29, 62, 63], and have shown better performance compared to traditional machine learning approaches [18]. Vasan et al. [62] propose to adopt transfer learning, which fine-tune a CNN network pre-trained with the ImageNet dataset [15], with the images of malware files to better conform the malware family classification task. The empirical results show that their architecture stands out among several deep learning models.

Image-based approaches require no domain-specific knowledge, and many existing well-developed image classification models can be used directly. However, they also have drawbacks. For example, transforming malware into images introduces new hyper-parameters (e.g., image width) and imposes non-existing spatial correlations between pixels in different rows which might be false [17].

2.2 Binary-Based Techniques

Binary-based approaches take the binary code of malware as input, which is considered as sequential information. Therefore, existing sequential models, especially from the Natural Language Processing (NLP) domain, are usually adopted for classification.

Jain et al. [26] propose to extract n–grams from the raw byte sequences as features, and leverage several basic machine learning models like Naïve Bayes [49] and AdaBoost [16] for malware detection. However, with the increase of n, the computation cost

Table 1: The Malware Family Classification Methods Studied

Category	Model	Related Work	Input Format for Model	PE Dataset for Evaluation
Image-based	VGG-16	Vasan et al. [62]	color image	Malimg
	ResNet-50	Vasan et al. [62]	color image	Malimg
		Singh et al. [54]	color image	Self-constructed dataset*
		Rezende et al. [48]	color image	Malimg
		Bhodia et al. [11]	grayscale image	Malicia, Malimg
	Inception-V3	Vasan et al. [62]	color image	Malimg
	IMCFN	Vasan et al. [62]	grayscale image, color image	Malimg
Binary-based	CBOW+MLP	Qiao et al. [43]	byte embedding ascending matrix	BIG-15
	MalConv Related	Raff et al. [44]	raw byte values	Closed dataset★
		Krcál et al. [32]	raw byte values	Closed dataset★
Disassembly-based	MAGIC	Yan et al. [65]	attribute control flow graph	BIG-15, YANCFG
	Word2Vec+KNN	Awad et al. [10]	opcode sequences	BIG-15
		Chandak et al. [13]	20 most frequent opcodes	Malicia, Self-constructed dataset*
	MCSC	Ni et al. [41]	opcode sequences Simhash image	BIG-15
		Kwon et al. [34]	opcode sequences Simhash image	BIG-15

* The malware samples of self-constructed dataset comes from various open source data release websites.
★ The dataset was provided by an anti-virus industry partner with whom the authors work, which cannot be accessed publicly.

increases exponentially. Raff et al. [44] propose the first end-to-end shallow CNN model that directly takes the entire binary file as input for malware classification. As a result, it will requires large memory for large malware files, and thus has limited processing capability. To solve the problem, the approach that focuses only on the PE-header of binaries [46] is proposed. It selects 328 bytes from the PE-header as inputs, and thus is not affected by the size of malware files. It is also shown to perform better than the compared approach that depends on domain knowledge features extracted by PortEX library [19]. Qiao et al. [43] treat each malware binary file as a corpus of 256 words (0x00-0xFF), adopt the Word2Vec model [38] to obtain the word embeddings, then represent the malware as a word embedding matrix in byte ascending order, and finally classify malware using Multi-Layer Perceptron (MLP).

Binary-based approaches do not require domain knowledge and consider the contextual information in malware binaries. However, representing a malware sample as a sequence of bytes may present some challenges compared to other category of methods. First, by treating each byte as a unit in a byte sequence, the size of the malware byte sequences may reach several million time steps [44], which is rather resource consuming. Second, adjacent bytes are expected be correlated spatially, which may not always hold due to jumps and function calls, and thus there might be discontinuities in the information within the binary files.

2.3 Disassembly-Based Techniques

Disassembly-based approaches first disassemble binary files into assembly code, and perform malware classification based on features such as Function Call Graph (FCG) and Control Flow Graph (CFG), that are extracted from assembly code.

Kinable et al. [30] propose to calculate the similarity score between two FCGs through existing graph matching techniques and use it as the distance metric for malware clustering. Kong et al. [31] present a generic framework that can first abstract malware into Attribute Function Call Graphs (AFCGs) and then further learn discriminant malware distance metrics to evaluate the similarity between two AFCGs. The above approaches are computation-intensive while calculating the similarity between graphs, which will bring huge performance overhead and cannot generalize well. Recently, Hassen et al. [20] propose to cluster similar functions of FCGs by using Minhash [12], and then represent the graphs as vectors for classification via leveraging deep learning models. Similarly, Yan et al. [65] employ Deep Graph Convolution Neural Network (DGCNN) [66] to aggregate the attributes on the AFCGs extracted from disassembly files, which is the first attempt for DGCNN on malware classification tasks.

There are also some approaches which extract features directly from assembly code. Specifically, the opcode sequence is usually adopted. Santos et al. [51] propose a feature representation method based on the frequency of the appearance of opcode sequence. Awad et al. [10] treat opcode sequences of each disassembly file as a document and apply the Word2Vec model [38] to generate a computation representation of the document. Finally, they use the Word Mover's Distance (WMD) [33] metric and K-Nearest Neighbour (KNN) to classify these documents. SimHash [41] and MinHash [57] adopt Hash projections to convert opcode sequences into vectors, which are then visualized into images for classification.

The disassembly-based techniques can better capture the code structure features compared to other methods, but they usually require domain knowledge, such as assembly language and its corresponding analysis methods.

3 DESIGN OF EMPIRICAL STUDY

3.1 Research Questions

RQ1: How do different PE malware family classification methods perform on different datasets? Although a large number of learning-based approaches are proposed to solve the malware family classification task, they are only evaluated independently with some specific dataset, e.g., public dataset BIG-15 [50], some manually crafted dataset [54] or dataset provided by their industry partners which are not public available [44]. There is a lack of systematic study to evaluate the performance of different approaches consistently on the same experiment settings with multiple datasets.
RQ2: How is the classification performance of various models affected by malware concept drift? Concept drift [28], which

is the change in the statistical properties of an object in unforeseen ways, is a realistic and critical issue in the PE malware family classification task. It is thus important to evaluate the performance of different approaches in the application scenario of concept drift.

RQ3: What factors hinder the current learning-based PE malware classification approaches from being deployed in industry and the corresponding improvement directions? With the gaps identified by the previous research questions, our ultimate goal is to provide suggestions on how to make the current learning-based PE malware classification approaches applicable in real industry scenarios.

3.2 Studied Methods

In order to systematically study the performance of different techniques, we select 9 state-of-the-art learning-based PE malware family classification methods, which cover image-based, binary-based and disassembly-based techniques, for our empirical study. Table 1 lists the details of all methods adopted.

3.2.1 Image-Based. **VGG-16:** VGGNet [53] was proposed to reduce the number of parameters in the convolution layers (with smaller convolution kernels) and improve on training time. VGG-16 is the most popular network architecture in the VGGNet family. The standard VGG-16 contains 13 Convolution layers (CONV), 5 max Pooling layers (Pool) and 3 Fully connected layers ($4096 \times 4096 \times N$, where N is decided by the number of predicted classes). All hidden layers use the Rectified Linear Unit (ReLU). For the output layer, softmax is used. The advantages of VGG-16 are its simple structure and its high fitting capability due to the large number of parameters.

ResNet-50: ResNet [21] is proposed to solve the problems of information loss, vanishing gradient and gradient explosion which commonly exist in the information transmission of traditional CNN networks when the network layers are deep. It protects the integrity of information by taking the input directly to the output in a structure called *skip connection (shortcut)*. Resnet-50 is a typical network in ResNet family. Because of the design of *shortcut*, ResNet-50 can train a deeper network which contains 50 hidden layers. ResNet-50 finally uses the global average Pool, and then uses softmax for the final prediction layer. ResNet-50 is more efficient than VGG-16 because it has far fewer parameters.

Inception-V3: InceptionNet [24, 58–60] aims to figure out how to use a dense component to approximate or replace an optimal locally sparse convolution structure. Four versions of InceptionNet have been developed, in which Inception-V3 [60] is the most representative one. It contains both symmetric and asymmetric building blocks, including CONV, average Pool, max Pool, Concatenation, dropout and FCs. Batch-normalization is widely used throughout the model. Like ResNet-50, Inception-V3 also ends up using a global average Pool, and then uses softmax for the final prediction layer. Inception-V3 was a first runner up in the ImageNet Large Visual Recognition Challenge (LVRC) where it outperformed VGGNet on error rate [14]. It is more effective compared to VGG-16 and it has a even smaller number of parameters than ResNet-50.

IMCFN: IMCFN stands for Image-based Malware Classification using Fine-tuned Convolutional Neural Network, which is customized to the task of malware family classification. It is a variant of VGG-16 that reduces the neurons in the first two FCs from 4,096 to 2,048, and adds a dropout layer to reduce the effect of overfitting.

Table 2: Details of MalwareBazaar dataset

Family Name	Number of Samples
Gozi	767
GuLoader	589
Heodo	214
IcedID	578
njrat	942
Trickbot	881
Total	3,971

Table 3: Details of MalwareDrift dataset

Family Name	Samples of Pre-drift	Samples of Post-drift
Bifrose	171	107
Ceeinject	90	458
Obfuscator	143	61
Vbinject	379	653
Vobfus	64	218
Winwebsec	218	269
Zegost	180	114
Total	1,245	1,880

3.2.2 Binary-Based. **Word2Vec+MLP:** This approach combines Word2Vec and the Muli-Layer Perception (MLP) model for malware family classification [43]. The key idea of this method is that the relationship of bytes in samples from the same family is similar, and is distinctly different from those samples of different families. Therefore, the vector matrix of raw bytes is a valid feature for malware classification. The raw binary is first pre-processed, removing 5 or more consecutive 0x00 or 0xCC (meaningless bytes). Each file is then taken as a corpus, which is considered to be composed of 256 words from 0x00 to 0xFF. The Continuous Bag-of-Word Model (CBOW) in Word2Vec [38] is used to obtain embedding vectors of 256 bytes in the file, and each file is represented as a byte vector ascending matrix. MLP takes these matrices as inputs and outputs the corresponding family categories. MLP consists of 3 FCs ($512 \times 512 \times N$, where N is decided by the number of predicted classes), and for the first two FCs, a dropout layer is added. MLP is the most intuitive and simplest deep neural network. Similar to VGG-16, although its structure is relatively simple, it has many parameters and can fit the training data well.

MalConv: This is the first end-to-end malware detection model that allows the entire malware to be taken as input. MalConv [44] first uses an embedding layer to map the raw bytes to a fixed 8 dimensional vector. In this way, it can captures high level location invariance in raw binaries by considering both local and global contexts. Then, MalConv use a shallow CNN with a large filter width of 500 bytes combined with an aggressive stride of 500. This allowed the model to better balance computational workload in a data-parallel manner using PyTorch [3] and thus can relieve the problem of the GPU memory consumption in the first convolution layer. As a shallow CNN architecture, MalConv conquers one of the primary practical limitations that reading the whole malware bytes is memory consuming, it also captures global location invariance in raw binaries. It allows the embedding layer to be trained jointly with the convolution layer for better feature extraction.

3.2.3 Disassembly-Based. **MAGIC:** This is an end-to-end malware detection method that classifies malware programs represented by

Attributed Control Flow Graph (ACFG) using Deep Graph Convolutional Neural Network (DGCNN) [66]. It first to convert the ACFG, which abstracts the vertices of the Control Flow Graph (CFG) as discrete numerical value vectors, extracted from malware disassembly file into a numerical vector. DGCNN then transforms these un-ordered ACFGs of varying sizes to tensors of fixed size and order for malware family classification.

Word2Vec+KNN: This method models malware disassembly file as a malware language, extracts the opcode sequence of it as malware document and uses the Word2Vec model to generate a computational representation of such document. This work choose Word Mover's Distance (WMD) [33] as the measure of semantic closeness between documents for KNN classification, which computes the cost of transporting all embedded words of document A to all embedded words of document B.

MCSC: It first extracts opcode sequences from disassembly files and encodes them to equal length based on SimHash [37]. Then, it takes each SimHash value as a binary pixel and converts the SimHash bits to grayscale images. It trains a CNN structure modified from LeNet-5 [35] to classify these grayscale images. When training the CNN classifier, muti-hash and bilinear interpolation are used to improve the accuracy of the model, and major block selection is used to decrease the image generation time.

3.3 Experimental Datasets

We employ four different datasets for evaluation, i.e., BIG-15 [50], Malimg [40], MalwareBazaar and MalwareDrift, respectively. The first two datasets are adopted from prior approaches, while the last two are newly constructed for our study. BIG-15 and Malimg are commonly used by previous research work. They are open-source and well-maintained with a large number of diverse families of malware. MalwareBazaar is constructed due to the reason that currently the only public available dataset to measure all methods discussed in Sec. 3.2 is BIG-15, which was released in 2015 and could have been dated. Therefore, we build a new dataset with the latest occurred malware to eliminate the experimental bias caused by using a single dataset; MalwareDrift is used to evaluate the effectiveness of existing approaches when facing the scenario of concept drift. We publish all above datasets used in the study for future research in this direction.

BIG-15 [50] is first released in the Malware Classification Challenge on Kaggle by Microsoft in 2015. This public dataset contains 10,868 labeled PE malware samples from 9 families. The raw malware are converted to binary and disassembly files for safety concerns.

Malimg [40] comprises of 9,435 malware collected from 25 families in 2011. This dataset only contains the greyscale images converted from the malware binary files and is widely used by image-based malware classification approaches.

MalwareBazaar is a new dataset we constructed according to the MalwareBazaar website [4], which provides free and unrestricted download services for malware samples. We first choose the top-6 malware families from the data released in 2020, and then download the most recently uploaded 1,000 malware samples for each family. Then, we filter out samples that are not in PE format and further leverage Joe Security [36] and AVClass [52] to check the label of each sample to remove noise samples, which different website give inconsistent labels. As a result, we obtain a dataset with 3,971 PE malware samples from 6 families, which is summarized in Table 2. To Make the dataset conform to different approaches, we further used IDA Pro [2] to convert the malware into binary and disassembly files. Please note that since MalwareBazaar is constructed from the latest malware samples, it can better reflect the tendency of the latest malware, which may provide a different perspective and complement with previous datasets, such as BIG-15.

MalwareDrift is constructed based on the conclusions by Wadkar et al. [64]. Their experiments confirmed that code changes appear as sharp spikes in the χ^2 timeline statistic, which quantifies the weight differences in Support Vector Machine (SVM) trained with PE malware features over the sliding time windows. Code changes can also be understood as the evolution or drift of a malware family. We adopt the dataset used in [64] as well as the corresponding χ^2 timeline statistic graph, based on which we can decide the evolution time period of each malware family according to the spikes, and divide samples of each family into the pre-drift and post-drift parts. After eliminating families without obvious sharp spikes and families of small sample size, we finally obtain the drift dataset, including 3,125 samples from 7 families, as is shown in Table 3.

3.4 Experimental Setup

3.4.1 Data Pre-processing. In order to conform to the requirement of different methods and provide a consistent experiment setting, we first perform data pre-processing for the adopted datasets. (1) For image-based methods, we first transform the malware files into the color image format as it has been shown that color images achieve better performance than greyscale images on the malware family classification task [62]. More concretely, we first transform the files into images of different width according to the file size, and then adopt the Nearest Neighbor interpolation image resize method [42] to resize the image into the size of 224*224. It has been shown that the original texture features remain sharp with this resize method [62]. (2) For the Word2Vec+MLP method, we follow the original settings [43] to remove all '0x00' and '0xcc' bytes that consecutively appear more than 5 times, and employ the Gensim library [47] for byte embedding. (3) For MalConv, we limit the malware file size to less than 2MB due to the reason that larger file sizes cause excessive GPU memory consumption [44]. (4) For MAGIC, we adopt IDA pro [2] to extract the Attributed Control Flow Graph (ACFG) for all malware files into documents. (5) For the Word2Vec+KNN method, we use the sed stream editor [6] written in shell script to convert all assembly code to x86-64 opcode sequences. (6) For MCSC method, we apply SimHash-768 [1] to convert the Opcode sequence into a SimHash sequence, and then convert it to a black and white image and utilize the bilinear interpolation to uniformly zoom the original image to the size of 32x32.

3.4.2 Configurations. In order to provide a fair comparison, we adopt the default hyper-parameter settings for the methods if they are released in the original paper [10, 41, 65], which are said to achieve the best performance of the corresponding methods. We only extract the opcode sequences in the Word2Vec+KNN approach [10] for the sake of time. For the other methods which do not report the hyper-parameters leading to the best performance,

Table 4: Performance of all methods on three datasets (10-fold cross-validation)

Dataset	Category	Model	Classification Performance (%)				Train Time (min)	Reasource Overhead		
			A	P_{macro}	R_{macro}	$F1_{macro}$		Mem (GB)	GPU.Mem (GB)	GPU (%)
BIG-15	Image	ResNet-50	**98.42**	**96.57**	**95.68**	**96.08**	60.13	36.86	10.77	85
		VGG-16	93.94	90.32	81.89	87.28	275.52	27.65	10.90	75
		Inception-V3	96.99	93.67	94.46	94.03	120.00	36.86	10.77	71
		IMCFN	97.77	95.93	94.81	95.13	52.32	58.37	10.77	95
	Binary	CBOW+MLP	**98.41**	**97.63**	**96.67**	**97.12**	1.46	13.31	10.44	29
		MalConv	97.02	94.34	92.62	93.33	227.40	259.07	10.79×4	37
	Disassembly	MAGIC	98.05	**96.75**	94.03	95.14	52.32	13.31	9.88	73
		Word2Vec+KNN	**98.07**	96.41	**96.51**	**96.45**	0.09	1.30	-	-
		MCSC	97.94	95.97	96.17	96.06	3.62	3.78	10.77	33
Malimg	Image	ResNet-50	98.14	95.48	95.06	95.21	140.02	32.77	10.77	95
		VGG-16	99.08	97.71	97.74	97.73	58.63	32.77	10.77	94
		Inception-V3	97.77	96.29	93.82	93.98	273.12	32.77	10.77	94
		IMCFN	**99.13**	**98.08**	**97.62**	**97.84**	34.95	43.01	10.77	96
MalwareBazaar	Image	ResNet-50	96.68	96.91	96.75	96.83	8.35	17.41	10.77	95
		VGG-16	96.35	96.58	96.54	96.56	43.96	18.43	10.77	97
		Inception-V3	95.83	95.67	95.79	95.73	6.39	12.29	10.77	96
		IMCFN	**97.38**	**97.53**	**97.41**	**97.47**	18.75	22.53	10.77	92
	Binary	CBOW+MLP	**97.81**	**97.92**	**98.08**	**98.00**	0.82	51.94	10.44	34
		MalConv	95.92	96.04	96.43	96.20	65.40	246.78	10.78	60
	Disassembly	MAGIC	92.82	88.03	87.36	87.45	246.00	113.66	10.61	81
		Word2Vec+KNN	95.64	93.34	94.29	93.79	≈0	5.54	-	-
		MCSC	**96.80**	**94.97**	**94.51**	**94.70**	1.07	45.34	10.77	33

we perform an intensive manual tuning process and employ the settings that achieve the best performance. Particularly, we apply the early stopping mechanism to fairly compare the learning efficiency for different methods. Due to the space limit, we do not report the detailed experiment settings in the paper, the information is available in our open-source repository.

As transfer learning is usually adopted for image-based methods in the malware family classification task to enhance the performance. We also evaluate the effect of transfer learning in our study. Following the standard process, we fix the structure (i.e., the number of layers and neurons per layer) of each model. We first perform pre-training with the ImageNet dataset [15] and then fine tune the models with our malware datasets in image format. To explore the effect of dataset size on performance, we use 10%, 50%, 80% and 100% of the malware data, respectively, for fine-tuning.

Particularly, to evaluate the performance of different methods in the concept drift scenario, we use the pre-drift samples to train a model following a standard data partition of 8:2 for training and testing, and report the performance of models on the pre-drift data. Then, we load the trained model and test it on the post-drift data.

Following the standard paradigm, we apply 10-fold cross-validation in our experimental comparison among different methods on different datasets, and 5-fold cross-validation on the concept drift experiment. We utilize the Macro Average Metrics of Precision (P_{macro}), Recall (R_{macro}) and F1-Score ($F1_{macro}$) to measure the multi-classification performance of a model on multiple malware families. Suppose there are N malware families (with a total of S samples) and we conduct M-fold cross-validation. First we calculate the total Precision (P_n), Recall (R_n) and F1-Score ($F1_n$) of M-fold cross-validation for each family class n ($1 \le n \le N$). The formulas used to calculate the metrics are shown in fomulas 1-3, where TP_{mn}, FP_{mn} and FN_{mn} represent the true positive, false positive and false negative of malware family n in the m-fold.

$$P_n = (\sum_{m=1}^{M} TP_{mn})/(\sum_{m=1}^{M}(TP_{mn} + FP_{mn})) \quad (1)$$

$$R_n = (\sum_{m=1}^{M} TP_{mn})/(\sum_{m=1}^{M}(TP_{mn} + FN_{mn})) \quad (2)$$

$$F1_n = (2 * P_n * R_n)/(P_n + R_n) \quad (3)$$

Based on the per-family precision (P_n), recall (R_n) and F1-score (F_n) ($1 \le n \le N$) computed with the M-fold data, the Macro Average of Precision, Recall and F1-score for multiple families are calculated with formula 5-7. We also compute the accuracy with formula 4.

$$A = (\sum_{n=1}^{N}\sum_{m=1}^{M} TP_{mn})/S \quad (4)$$

$$P_{macro} = (\sum_{n=1}^{N} P_n)/N \quad (5)$$

$$R_{macro} = (\sum_{n=1}^{N} R_n)/N \quad (6)$$

$$F1_{macro} = (\sum_{n=1}^{N} F1_n)/N \quad (7)$$

3.4.3 Executing Environment. All of our experiments are conducted on a server with 2 Intel Xeon Platinum 8260 CPU @2.30GHz, 8 Nvidia GeForce RTX2080 Ti GPU (11GB memory), and 512GB RAM.

4 EXPERIMENTAL RESULT AND ANALYSIS

4.1 RQ1: Performance Comparison

Table 4 shows the experimental results of all adopted models on the three studied datasets. Particularly, Malimg was only used for image-based models as the original malware file is unavailable.

From the table we can see that the method with CBOW+MLP model achieves the best performance in terms of F1-score compared with all the others on both BIG-15 and MalwareBazaar datasets, indicating the generalizability of it to conform different datasets. On the other hand, when considering the performance of different categories, there is no individual category that always outperforms the others. For example, for binary-based methods, though the

Table 5: Performance of transfer learning for image-based approaches on three datasets (10-fold cross-validation)

Dataset	Model	Training Strategy	Classification Performance (%)				Train Time (min)	Reasource Overhead		
			A	P_{macro}	R_{macro}	$F1_{macro}$		Mem (GB)	GPU.Mem (GB)	GPU (%)
BIG-15	ResNet-50	10% TL	94.76	92.74	82.69	86.77	3.97	22.53	10.56	3
		50% TL	96.68	95.35	85.22	91.50	24.78	22.53	10.56	3
		80% TL	96.90	94.87	85.53	92.50	37.02	23.55	10.56	3
		100% TL	**97.05**	**95.64**	**93.60**	**94.48**	43.62	21.50	10.56	4
	VGG-16	10% TL	89.56	78.74	76.73	77.59	0.42	19.46	10.44	3
		50% TL	95.21	89.02	83.77	87.21	11.76	21.50	10.44	44
		80% TL	96.62	92.00	85.20	90.41	8.04	22.53	10.44	71
		100% TL	**96.78**	**93.36**	**91.78**	**92.46**	9.72	23.55	10.44	70
	Inception-V3	10% TL	92.28	91.46	80.51	80.94	3.27	18.43	10.56	4
		50% TL	95.10	91.17	86.76	86.87	12.78	19.46	10.56	8
		80% TL	95.31	92.52	88.54	88.64	19.02	19.46	10.56	8
		100% TL	**95.56**	**93.48**	**89.64**	**89.76**	22.92	21.50	10.56	17
	IMCFN	10% TL	94.81	94.13	84.04	85.21	4.52	20.48	10.44	25
		50% TL	97.38	95.25	91.90	93.15	14.13	22.53	10.44	34
		80% TL	97.81	96.09	93.69	94.69	21.04	28.67	10.44	40
		100% TL	**98.05**	**96.18**	**94.37**	**95.18**	21.41	32.77	10.44	40
Malimg	ResNet-50	10% TL	97.34	93.79	92.77	93.21	5.29	19.46	10.56	7
		50% TL	98.02	95.61	94.46	94.93	29.61	19.46	10.56	5
		80% TL	**98.59**	**96.37**	**96.23**	**96.29**	44.65	19.46	10.56	8
		100% TL	98.49	96.33	95.88	96.10	70.80	20.48	10.56	7
	VGG-16	10% TL	96.37	90.05	89.24	89.50	2.40	17.41	10.44	7
		50% TL	97.24	91.29	91.64	91.41	7.32	16.38	10.44	5
		80% TL	98.58	96.35	96.23	96.28	11.46	17.41	10.44	20
		100% TL	**98.63**	**96.47**	**96.40**	**96.42**	14.16	19.46	10.44	52
	Inception-V3	10% TL	93.82	87.07	83.93	85.05	3.93	16.38	10.56	6
		50% TL	95.75	89.23	88.63	88.86	11.04	17.41	10.56	2
		80% TL	96.17	90.15	89.61	89.84	16.32	20.48	10.56	12
		100% TL	**96.43**	**90.62**	**90.28**	**90.41**	18.96	18.43	10.56	7
	IMCFN	10% TL	91.48	74.18	74.05	73.35	4.63	18.43	10.44	26
		50% TL	96.67	90.70	90.06	90.27	12.66	22.52	10.44	35
		80% TL	96.96	91.17	90.85	90.93	18.41	22.53	10.44	39
		100% TL	**97.13**	**91.29**	**91.33**	**91.25**	18.55	23.55	10.44	40
MalwareBazaar	ResNet-50	10% TL	90.03	89.48	90.99	90.09	4.07	9.14	10.56	6
		50% TL	93.98	94.28	94.57	94.38	7.23	9.13	10.56	7
		80% TL	94.90	95.21	95.34	95.25	9.11	9.16	10.56	5
		100% TL	**95.30**	**95.66**	**95.68**	**95.65**	10.01	9.19	10.56	6
	VGG-16	10% TL	83.55	86.50	76.12	76.80	1.16	8.17	10.44	56
		50% TL	92.35	92.75	92.54	92.51	2.19	9.34	10.44	52
		80% TL	93.85	93.92	94.37	94.06	2.98	10.24	10.44	55
		100% TL	**94.43**	**94.43**	**94.90**	**94.60**	3.94	11.26	10.44	52
	Inception-V3	10% TL	90.28	90.87	90.99	90.87	0.79	7.83	10.56	7
		50% TL	93.75	94.31	94.20	94.24	3.77	8.13	10.56	8
		80% TL	94.48	94.96	94.87	94.90	6.37	8.70	10.56	6
		100% TL	**94.80**	**95.15**	**95.17**	**95.15**	8.47	8.47	10.56	7
	IMCFN	10% TL	91.30	91.72	91.27	91.35	2.14	9.39	10.44	27
		50% TL	94.95	94.99	95.33	95.10	5.75	10.24	10.44	37
		80% TL	95.55	95.59	95.86	95.68	6.25	11.26	10.44	45
		100% TL	**95.73**	**95.73**	**96.02**	**95.84**	7.76	11.26	10.44	37

CBOW+MLP model performs best, the MalConv model is not always better than the other methods. Base on this result, we can conclude that the data format of malware should not be a critical factor that impacts the classification performance, which is more likely decided by the model itself. Besides, by comparing the experimental results across different datasets for each model, most of models can achieve stable performance except VGG-16 and MAGIC, whose F1-scores vary greatly on the BIG-15 and MalwareBazzar datasets. We further analyze the results and observe that imbalanced data is a major reason for VGG-16. For example, it performs much worse on BIG-15 compared with other datasets, because there are only 42 samples belonging to the Simda family, which only accounts for 1.43%-10.55% of the other families and less than 0.4% of the total dataset. Therefore, VGG-16 can hardly learn the deep semantics from such limited samples as it has the largest number of parameters (see Table 6). While for MAGIC, the reason is that it has large runtime GPU memory consumption, especially when the input file size is large. For BIG-15, the largest disassembly file is 140MB, while for MalwareBazaar, the file size can exceed 1GB. We can only set the batch size to 1 when processing the MalwareBazaar dataset due to the limitations of our GPU memory. The small batch size could affect the performance of the method.

Considering each individual category, though IMCFN in general achieves the best performance (average F1-score: 96.81%) compared with the other image-based methods, there is no evidence that it significantly outperforms the others. Similar to CBOW+MLP (average F1-score: 97.56%) and MCSC (average F1-score: 95.38%) for the binary-based and disassembly-based methods. The overall performance within each category is relatively close to each other.

Finding 1. *The binary-based model CBOW+MLP performs the best across different datasets among all methods, while no individual category significantly outperforms the others.*

Table 6: The resource consumption of various methods in runtime

Category	Model	Time		Saved Model Size (MB)				Runtime memory (MB)
		Pre-process (s)	Predict (ms)	Save Weights Only		Save Entire Model		
				From Scratch	Transfer	From Scratch	Transfer	
Image	ResNet-50	0.7	2.62	90.27	90.45	270.51	180.94	2359.50
	VGG-16		2.21	512.36	512.36	1024.00	1024.73	2312.17
	Inception-V3		2.03	83.91	83.91	250.77	168.04	2369.00
	IMCFN		2.23	268.28	268.28	536.57	536.57	2299.00
Binary	CBOW+MLP	1.09	5441.02	129.03		258.07		1406.67
	MalConv	0.28	14.02	3.96		7.91		2378.50
Disassembly	MAGIC	17.78	23.63	412.14		1239.04		3898.83
	Word2Vec+KNN	17.72	95243.31★	-		-		-
	MCSC	4.49	3477.77	3.63		7.27		2275.67

We report the average time taken to process and predict one sample; ★: Contains the time to calculate 6893 document distances;–: KNN cannot generate a model that can be saved; The sizes reported on transfer learning contain the parameters of the pre-trained feature extraction layers, the fine-tuned classification layer and the model structure (for the entire model setting only).

It has been demonstrated above that insufficient training data for large-scale networks (e.g., VGG-16) may cause big performance drop. Transfer learning has been widely adopted in the image processing domain to tackle similar issues and obtained good performance, and thus we also investigate its impact on PE malware classification task in this work. We utilize the pre-trained model on ImageNet [15] and use the corresponding malware image data to fine tune the last fully-connected layer. The experimental results are shown in Table 5. Specifically, VGG-16, which did not perform well due to insufficient training data, was significantly improved from 87.28% to 92.46% on BIG-15 through transfer learning. However, according to the results, the impact of transfer learning was still limited and the performance may largely drop for some cases, e.g., IMCFN. As a result, we further employ a more aggressive strategy that permits to update the connection weights in the last representation layer during fine-tuning. Figure 1 shows results of the IMCFN model. From the figure, we can observe that opening both the fully connected layer and the convolution layers in general leads to better performance, especially on Malimg. We also measure the cost of fine-tuning the last fully-connected layer only and the aggressive strategy, where we find that the latter causes 1.5x-3x training time (less than 1 hour) compared with the former, yet gains an average of 4.14% performance increase in terms of F1-score. More importantly, the aggressive strategy exactly advances the classification performance, in terms of F1-score , by 0.51%-2.94% compared with training from scratch, except the Malimg dataset: 97.59% (aggressive) and 97.84% (from scratch).

The results indicate that converted malware images are different, in terms of image features, from typical images in ImageNet. Therefore, the features trained with ImageNet are not directly applicable to malware images, and thus we need to open the feature extraction layers (i.e., the convolution layers in the models we studied) in the fine-tuning process for a better performance.

Finding 2. *Transfer learning can potentially further improve the effectiveness of image-based PE malware classification methods, and internal feature extraction layers of the model should be opened for fine-tuning to achieve better transfer-learning performance.*

Besides effectiveness, another important factor that impacts the practicality of PE malware classification methods in industrial scenarios is the efficiency and requirement of hardware environment

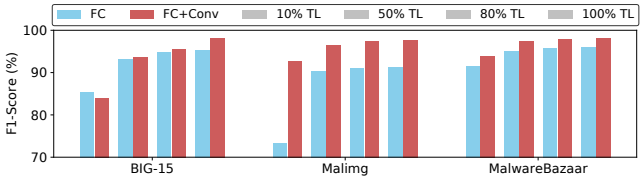


Figure 1: The F1-score of only open fully-connected layers (FC) vs. the aggressive strategy (FC+Conv) of IMCFN.

since some methods may be required to work on resource-critical devices, such as a network gateway device with limited computing resources. Table 4 and Table 5 show the time and consumption of resources in detail for each method during training of the corresponding models, and the average runtime resource consumption of each method is shown in Table 6. From the tables we can see that image-based methods are CPU intensive compared to the other categories, while transfer learning is a possible solution as it can help to reduce 28.49%∼96.47% training time. In addition, according to Table 6, image-based methods take very small pre-process and prediction overhead. The model sizes are relatively larger than the other methods. Disassembly-based methods take relatively longer preprocessing time, due to the time-consuming disassemble process. Particularly, the prediction time for the Word2Vec+KNN method is much longer, because it requires computing the distances between the given input data with all samples in the training set, all such factors indeed restrict the application of existing methods in the industry, and should be taken into consideration in future research.

4.2 RQ2: Performance under Concept Drift

Table 7 shows the evaluation results of different methods in the concept drift scenario. In particular, we choose the IMCFN as a representation of the image-based approaches as it shows the best overall performance (highest average F1-score) from the previous research question. In the table, we visualize the impact of concept drift for different methods with gradient color, the deeper the red color is, the higher the score drops.

From Table 7, we can observe that all existing methods suffer from a large performance drop while confronting the concept drift in real industry scenarios, where the reduction of F1-score is up to 27.07%-69.62%. The F1-scores for all methods on the post-drift dataset are no more than 45%, reflecting that existing methods fail

Table 7: Impact of Concept Drift on Performance

Category	Model	Drift Part	Classification Performance (%)			
			A	P	R	F
Image	IMCFN	pre-drift	85.21	85.23	83.00	**83.91**
		post-drift	49.90	52.74	44.42	42.10
		decrease	35.31	32.49	38.58	41.81
Binary	CBOW+MLP	pre-drift	81.69	83.36	78.58	80.50
		post-drift	17.44	11.52	14.45	10.88
		decrease	64.25	71.84	64.13	**69.62**
	MalConv	pre-drift	77.43	78.49	74.97	76.19
		post-drift	46.50	39.19	39.49	35.51
		decrease	30.93	39.30	35.48	40.68
Disassembly	MAGIC	pre-drift	80.85	81.29	77.77	79.30
		post-drift	42.15	34.69	33.07	28.30
		decrease	38.70	46.60	44.70	51.00
	Word2Vec+KNN	pre-drift	81.85	80.77	79.79	80.12
		post-drift	49.18	48.73	51.80	**43.87**
		decrease	32.67	32.04	27.99	36.25
	MCSC	pre-drift	74.31	69.44	73.49	70.89
		post-drift	50.58	46.97	48.25	43.82
		decrease	23.73	22.47	25.24	**27.07**

to consider the scenario of concept drift and there is still much improvement space. By comparing the results on concept-drift and non-concept-drift datasets, we can see that it is vital to take concept drift scenario into consideration for method evaluation.

Finding 3. *All existing methods suffer from poor performance when facing concept drift from real industrial scenarios, and thus they should be seriously considered when evaluating PE malware family classification methods.*

According to the results, the stability of different methods also varies greatly. The method of CBOW+MLP, though performs best when used in the common machine learning scenario, has the sharpest decrease in the concept drift scenario, which is mainly due to its simple structure. On the contrary, MCSC and Word2Vec+KNN show the least decrease ratios. The reason is that both of them extract Opcode sequences from disassembly files, and focus on the local context connection of Opcode sequences which tends to be retained during malware evolving and thus those methods show stable performance on concept drift. An interesting finding is that though Word2Vec+KNN also has simple structures, it performs better than CBOW+MLP under concept drift, which credits to the KNN algorithm that computes the similarity distances between the coming sample with all others. As a result, it typically requires a much longer prediction time (see Table 6).

Finding 4. *CBOW+MLP shows the worst performance to concept drift, while MCSC is the most robust to concept drift. Results indicate that Opcode sequence preserves the family feature of PE malware and thus are effective and robust features for classification under concept drift.*

In order to investigate whether there is a significant difference on different PE malware families, we present the results in Figure 2. We can observe that all methods tend to perform consistently on pre-drift dataset, with an exception of Obfuscator due to code obfuscations, which disturbs the feature extraction and prediction performance of different methods. However, different methods show diverse performance on the post-drift dataset and this is especially obvious for the Ceeinject and Vobfus families. Moreover, all methods show consistent small performance drop on Vbinject. By a close

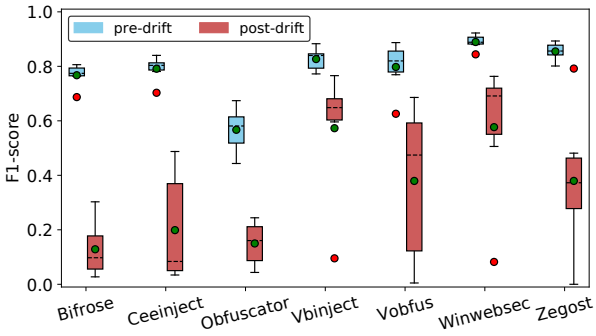
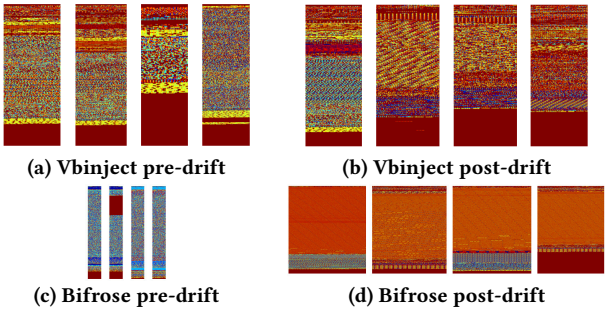


Figure 2: The F1-score box-plot of the Pre-drift and the Post-drift samples for all methods on the MalwareDrift dataset.



(a) Vbinject pre-drift **(b) Vbinject post-drift**

(c) Bifrose pre-drift **(d) Bifrose post-drift**

Figure 3: The visualized images of the Pre-drift and Post-drift samples on the Vbinject and Bifrose family

look into the data, we find the major reason is that samples after concept drift in Vbinject do not show sharp changes compared to other families. For example, Figures 3a and 3b show the visualized image of samples before and after the concept drift from Vbinject (named Group-1), while Figures 3c and 3d show the samples from Bifrose (named Group-2), which obtains the largest performance drop. Comparing the images, we can observe that the impact of concept drift is relatively smaller in Group-1 than in Group-2. We further use the difference Hash similarity [27] to measure the similarity between images before and after concept drift. The smaller the similarity value is, the more similar the images are. Finally, they are 26 and 32, respectively for Group-1 and Group-2, which indeed causes the big performance difference on different families.

4.3 RQ3: Factors that Affect the Industry Usage of the Current Approaches

The ultimate goal of our work is to encourage deployment of the research models in real industry scenarios. Therefore, we conduct an interview with our industry partner, who is in charge of the virus detection product of a security company. In particular, we ask the following questions based on our study results.

- What classification methods are currently adopted in the company and why.
- What are the factors that affect choosing the suitable methods in real applicable scenario?
- Is concept drift common in real application scenarios and what are the current status of handling concept drift?

Currently, **two mainstream approaches are adopted in industry application scenario, i.e., the sandbox and the pattern based approaches.** Sandbox can be described as a virtual environment which execute the malware and extract runtime feature for classification. It is accurate yet is time and resource consuming. For instance, the sandbox can process around 5-10 samples per minute, which can be tolerated due to its high accuracy. Pattern based approaches are static detection methods which are based on the pattern/feature database. It is efficient in terms of time and resource consumption, yet is fragile to noises, obfuscation and concept drift.

Industry usage of the PE malware classification methods are mainly limited by three factors, i.e., the prediction precision and recall, the predicting time as well as the resource consumption, and the main resource concerns are runtime memory and CPU usage. The first two factors decide the user experience and thus whether the corresponding method can be adopted, and the resource consumption decides what kind of devices can the methods be deployed on. As a concrete example, in one of their product which contains the learning-based malware classification model, they require the runtime memory to be below 1GB, which cannot be met by all of our studied methods. Another requirement is to be able to predict a malware within 0.1s with an accuracy above 93%, which fillters out most of the binary-based and disassembly-based methods.

Concept drift usually happens due to malware evolving, e.g., in scenarios where existing malware wants to escape detection, and there could be new non-kernel functionalities such as the communication and message passing techniques being changed. This happens frequently and raises challenges to malware family classification. There is a lack of specific mechanism to handle this case and current practice usually use the sandbox methods for such scenarios. Another observation is that in addition to concept drift, they also need to tackle with the challenge of the fast evolving new malware families and features. Except for the fine-grained family classification as defined by the existing academic datasets such as BIG-15, our industry partners are more interested in the detection of malware families based on their malicious behaviors, i.e., Trojan, Rootkit or Ransomware. However, there is a lack of research on this direction, highly likely due to the fact that there is no such datasets available.

Finding 5. *Real industry application scenario requires the classification methods to be able to tackle the challenges brought by the fast evolving of malware families. Moreover, there should be a trade-off between the resource consumption and prediction accuracy in consideration of the deployment environment and customer feedback. Therefore, the future research should focus more on (1) how to handle the fast evolving of malware family rather than only evaluate with one or a few dataset; (2) a more light-weight model with high prediction accuracy, and (3) malware family classification from the malicious behavior perspective.*

5 THREATS TO VALIDITY

Threats to internal validity mainly lie in the implementation of different methods. In order to compare the results fairly, we employ the reported best configurations for each model if available, while for others we report the best results we have obtained after an intensive manual tuning process. We believe this strategy will mitigate the bias involved by different model settings. Besides, due to the limit of physical memory, we set the batch size to 32 (originally 256) when training MalConv model, which may affect the results. However, we argue that it is reasonable and acceptable, especially in real application scenarios, where resources are critical according to the feedback from industry. In addition, we publish all our experimental data for replication and boosting future research.

Threats to external validity mainly lie in the selection bias of methods and datasets studied. In order to perform a systematic comparison, we have adopted 9 different methods, covering the mainstream image-based, binary-based and disassembly-based techniques. To alleviate the impact of datasets, we employ two commonly-used datasets (i.e., BIG-15 and Malimg), and further construct two new datasets to reflect the latest progress of PE malware (MalwareBazaar) and the concept drift issue (MalwareDrift) from real industry practice.

Threats to construction validity mainly lie in the randomness and measurements in our experiment. To reduce the impact of randomness, we applied a 10-fold cross-validation in our comparing experiments on BIG-15, Malimg and MalwareBazaar datasets, and used 5-fold cross-validation in studying the effects of concept drift on various classification methods, rather than repeating each experiment several times. Macro-average is widely adopted to measure the performance of multi-class classification.

6 CONCLUSION

PE malware family classification has gained great attention and a large number of approaches have been proposed. In this paper, we first identify the gap of applying learning-based PE malware classification approaches in industry through a systematic empirical study, where we employ 9 different methods, covering the mainstream image-based, binary-based and disassembly-based techniques, and 4 different datasets for the experiment. Based on the obtained quantitative evaluation results in Section 4.1, 4.2, and the requirements from industry (Section 4.3), we conclude that: (1) There is no individual class of methods significantly outperforms the others; (2) All class of methods show performance degradation on concept drift, which is vital important in practice; (3) The prediction time and high memory consumption hinder existing approaches from being adopted for industry usage. We further provide actionable guidance on future applied research: (1) focus more on how to handle the fast evolving of malware family; (2) explore light-weight models with high prediction accuracy; (3) take into account the malicious behavior features for malware family classification.

ACKNOWLEDGMENTS

This work is partially supported by the National Key Research and Development Program of China No. 2019QY1302; the NSFC-General Technology Basic Research Joint Funds under Grant U1836214; The NSFC Youth Funds under Grant 61802275; State Key Laboratory of Communication Content Cognition fund No. A32001.

REFERENCES

[1] 2009. *Secure hash and message digest algorithm library.* Retrieved March 2, 2021 from https://pypi.org/project/hashlib/

[2] 2017. *IDA 7.0.* Retrieved March 2, 2021 from https://www.hex-rays.com/products/ida/news/

[3] 2020. *PyTorch.* Retrieved April 20, 2021 from https://pytorch.org/

[4] 2021. *MalwareBazaar.* Retrieved May 1, 2021 from https://bazaar.abuse.ch/

[5] 2021. *March 1st – Threat Intelligence Report.* Retrieved April 20, 2021 from https://research.checkpoint.com/2021/march-1st-threat-intelligence-report/

[6] 2021. *sed, a stream editor.* Retrieved March 2, 2021 from https://www.gnu.org/software/sed/manual/sed.html

[7] 2021. *Top File Types | Statistics of MalwareBazaar.* Retrieved May 1, 2021 from https://bazaar.abuse.ch/statistics/

[8] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*, Elisa Bertino, Ravi S. Sandhu, and Alexander Pretschner (Eds.). ACM, 183–194. https://doi.org/10.1145/2857705.2857713

[9] AV-TEST. 2021. *Malware Statistics and Trends Report by AV-TEST.* Retrieved March 2, 2021 from https://www.av-test.org/en/statistics/malware/

[10] Yara Awad, Mohamed Nassar, and Haïdar Safa. 2018. Modeling Malware as a Language. In *2018 IEEE International Conference on Communications, ICC 2018, Kansas City, MO, USA, May 20-24, 2018*. IEEE, 1–6. https://doi.org/10.1109/ICC.2018.8422083

[11] Niket Bhodia., Pratikkumar Prajapati., Fabio Di Troia., and Mark Stamp. 2019. Transfer Learning for Image-based Malware Classification. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ForSE,*. INSTICC, SciTePress, 719–726. https://doi.org/10.5220/0007701407190726

[12] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, Bruno Carpentieri, Alfredo De Santis, Ugo Vaccaro, and James A. Storer (Eds.). IEEE, 21–29. https://doi.org/10.1109/SEQUEN.1997.666900

[13] Aniket Chandak, Wendy Lee, and Mark Stamp. 2021. A Comparison of Word2Vec, HMM2Vec, and PCA2Vec for Malware Classification. In *Malware Analysis Using Artificial Intelligence and Deep Learning*. 287–320. https://doi.org/10.1007/978-3-030-62582-5_11

[14] Jongwon Chang, Jisang Yu, Taehwa Han, Hyukjae Chang, and Eunjeong Park. 2017. A method for classifying medical images using transfer learning: A pilot study on histopathology of breast cancer. In *19th IEEE International Conference on e-Health Networking, Applications and Services, Healthcom 2017, Dalian, China, October 12-15, 2017*. IEEE, 1–4. https://doi.org/10.1109/HealthCom.2017.8210843

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[16] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139. https://doi.org/10.1006/jcss.1997.1504

[17] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications* 153 (2020), 102526. https://doi.org/10.1016/j.jnca.2019.102526

[18] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. 2019. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques* 15, 1 (2019), 15–28. https://doi.org/10.1007/s11416-018-0323-0

[19] Katja Hahn. 2014. Robust static analysis of portable executable malware. *HTWK Leipzig* (2014), 134.

[20] Mehadi Hassen and Philip K. Chan. 2017. Scalable Function Call Graph-based Malware Classification. In *Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017*, Gail-Joon Ahn, Alexander Pretschner, and Gabriel Ghinita (Eds.). ACM, 239–248. https://doi.org/10.1145/3029806.3029824

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[22] M. Howard, A. Pfeffer, M. Dalai, and M. Reposa. 2017. Predicting signatures of future malware variants. In *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*. 126–132. https://doi.org/10.1109/MALWARE.2017.8323965

[23] Giacomo Iadarola., Fabio Martinelli., Francesco Mercaldo., and Antonella Santone. 2020. Image-based Malware Family Detection: An Assessment between Feature Extraction and Classification Techniques. In *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security - AI4EIoTs ,*. INSTICC, SciTePress, 499–506. https://doi.org/10.5220/0009817804990506

[24] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, 448–456.

[25] Mugdha Jain, William Andreopoulos, and Mark Stamp. 2020. Convolutional neural networks and extreme learning machines for malware classification. *Journal of Computer Virology and Hacking Techniques* 16, 3 (2020), 229–244. https://doi.org/10.1007/s11416-020-00354-y

[26] Sachin Jain and Yogesh Kumar Meena. 2011. Byte level n–gram analysis for malware detection. In *International Conference on Information Processing*. Springer, 51–59. https://doi.org/10.1007/978-3-642-22786-8_6

[27] Huang Jiaheng, Li Xiaowei, Chen Benhui, and Yang Dengqi. 2017. A comparative study on image similarity algorithms based on hash. *Journal of Dali University* 2, 12, Article 32 (2017), 32–37 pages.

[28] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 625–642.

[29] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil DB Bruce, Yang Wang, and Farkhund Iqbal. 2018. Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 1–5. https://doi.org/10.1109/NTMS.2018.8328749

[30] Joris Kinable and Orestis Kostakis. 2011. Malware classification based on call graph clustering. *Journal in computer virology* 7, 4 (2011), 233–245. https://doi.org/10.1007/s11416-011-0151-y

[31] Deguang Kong and Guanhua Yan. 2013. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1357–1365. https://doi.org/10.1145/2487575.2488219

[32] Marek Krčál, Ondřej Švec, Martin Bálek, and Otakar Jašek. 2018. Deep convolutional malware classifiers can learn from raw executables and labels only. (2018).

[33] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *International conference on machine learning (Proceedings of Machine Learning Research, Vol. 37)*. PMLR, Lille, France, 957–966. http://proceedings.mlr.press/v37/kusnerb15.html

[34] Young-Man Kwon, Jae-Ju An, Myung-Jae Lim, Seongsoo Cho, and Won-Mo Gal. 2020. Malware Classification Using Simhash Encoding and PCA (MCSP). *Symmetry* 12, 5 (2020), 830. https://doi.org/10.3390/sym12050830

[35] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[36] Joe Security LLC. 2011. *Joe Security.* Retrieved May 1, 2021 from https://www.joesecurity.org/

[37] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*. 141–150. https://doi.org/10.1145/1242572.1242592

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[39] Barath Narayanan Narayanan, Ouboti Djaneye-Boundjou, and Temesguen M Kebede. 2016. Performance analysis of machine learning and pattern recognition algorithms for malware classification. In *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*. IEEE, 338–342. https://doi.org/10.1109/NAECON.2016.7856826

[40] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. 2011. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*. 1–7. https://doi.org/10.1145/2016904.2016908

[41] Sang Ni, Quan Qian, and Rui Zhang. 2018. Malware identification using visualization images and deep learning. *Computers & Security* 77 (2018), 871–885. https://doi.org/10.1016/j.cose.2018.04.005

[42] J Anthony Parker, Robert V Kenyon, and Donald E Troxel. 1983. Comparison of interpolating methods for image resampling. *IEEE Transactions on medical imaging* 2, 1 (1983), 31–39. https://doi.org/10.1109/TMI.1983.4307610

[43] Yanchen Qiao, Bin Zhang, and Weizhe Zhang. 2020. Malware Classification Method Based on Word Vector of Bytes and Multilayer Perception. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6. https://doi.org/10.1109/ICC40277.2020.9149143

[44] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435* (2017).

[45] Edward Raff and Charles Nicholas. 2020. A Survey of Machine Learning Methods and Challenges for Windows Malware Classification. *arXiv preprint arXiv:2006.09271* (2020).

[46] Edward Raff, Jared Sylvester, and Charles Nicholas. 2017. Learning the pe header, malware detection with minimal domain knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 121–132. https://doi.org/

10.1145/3128572.3140442

[47] Radim Rehurek. 2020. *gensim 3.8.3.* Retrieved March 2, 2021 from https://pypi.org/project/gensim/

[48] Edmar Rezende, Guilherme Ruppert, Tiago Carvalho, Fabio Ramos, and Paulo De Geus. 2017. 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, 1011–1014. https://doi.org/10.1109/ICMLA.2017.00-19

[49] Irina Rish et al. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3. 41–46.

[50] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft malware classification challenge. *arXiv preprint arXiv:1802.10135* (2018).

[51] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G Bringas. 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences* 231 (2013), 64–82. https://doi.org/10.1016/j.ins.2011.08.020

[52] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*. 230–253. https://doi.org/10.1007/978-3-319-45719-2_11

[53] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[54] Ajay Singh, Anand Handa, Nitesh Kumar, and Sandeep Kumar Shukla. 2019. Malware classification using image representation. In *International Symposium on Cyber Security Cryptography and Machine Learning*. Springer, 75–92. https://doi.org/10.1007/978-3-030-20951-3_6

[55] SonicWall. 2020. *2020 SonicWall Cyber Threat Report.* Retrieved March 2, 2021 from https://www.sonicwall.com/medialibrary/en/infographic/infographic-2020-sonicwall-mid-year-cyber-threat-report.pdf

[56] Alireza Souri and Rahil Hosseini. 2018. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences* 8, 1 (2018), 1–22. https://doi.org/10.1186/s13673-018-0125-x

[57] Guosong Sun and Quan Qian. 2018. Deep learning and visualization for identifying malware families. *IEEE Transactions on Dependable and Secure Computing* (2018). https://doi.org/10.1109/TDSC.2018.2884928

[58] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. 4278–4284.

[59] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

[61] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019), 123–147. https://doi.org/10.1016/j.cose.2018.11.001

[62] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng. 2020. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks* 171 (2020), 107138. https://doi.org/10.1016/j.comnet.2020.107138

[63] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Qin Zheng. 2020. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security* 92 (2020), 101748. https://doi.org/10.1016/j.cose.2020.101748

[64] Mayuri Wadkar, Fabio Di Troia, and Mark Stamp. 2020. Detecting malware evolution using support vector machines. *Expert Systems with Applications* 143 (2020), 113022. https://doi.org/10.1016/j.eswa.2019.113022

[65] Jiaqi Yan, Guanhua Yan, and Dong Jin. 2019. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 52–63. https://doi.org/10.1109/DSN.2019.00020

[66] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.